

AD-A055 763

ILLINOIS INST OF TECH CHICAGO DEPT OF COMPUTER SCIENCE
PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED L--ETC(U)
MAY 78 A S WOJCIK, R SWARTWOUT

F/G 12/1

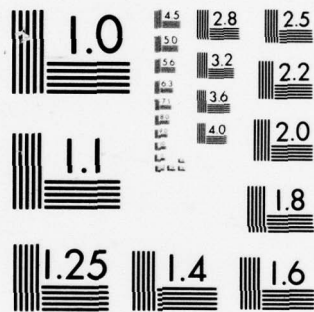
N00014-78-G-0019

NL

UNCLASSIFIED

1 OF 4
AD
A055763





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

FOR FURTHER TRANSMISSION

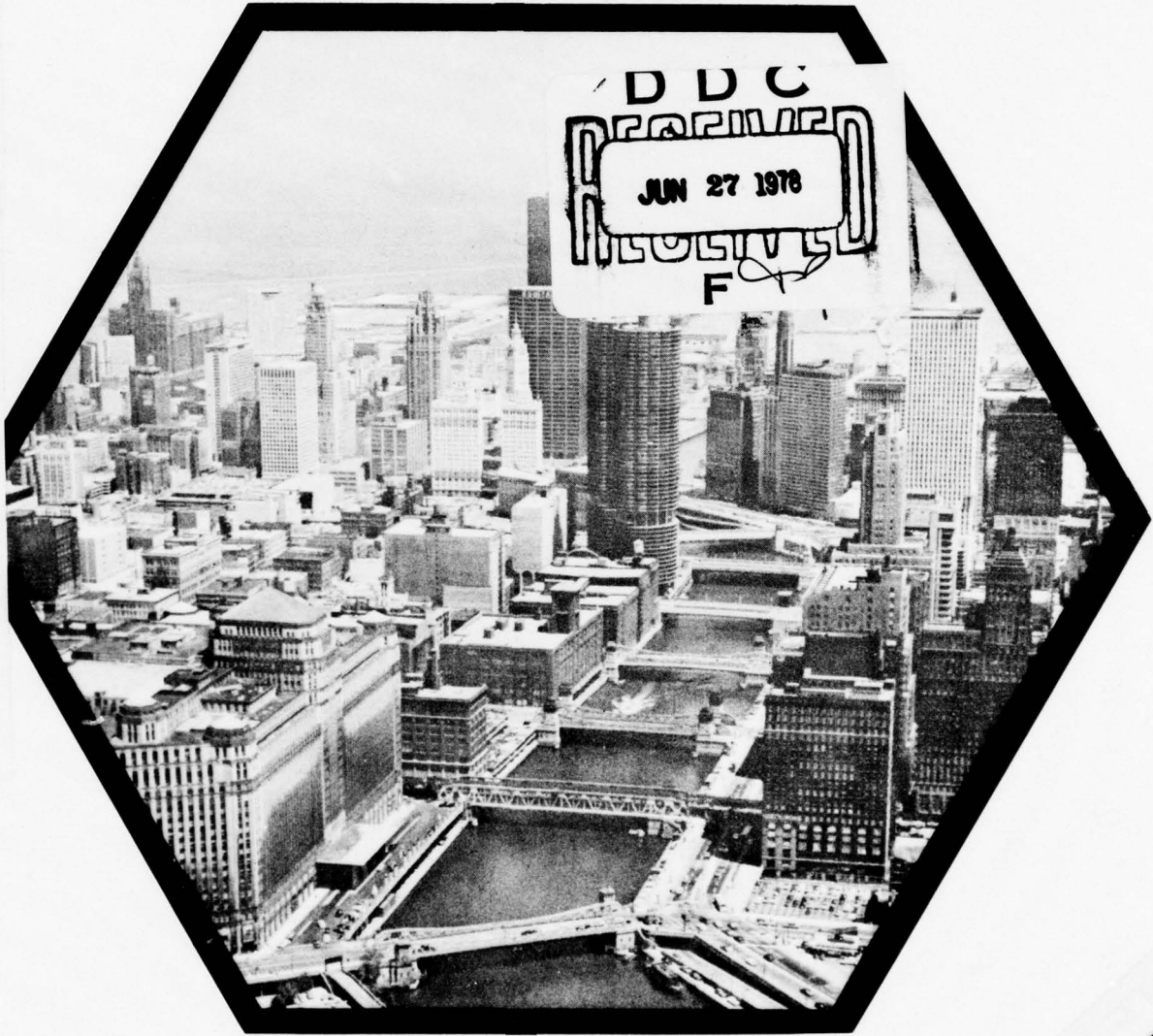


PROCEEDINGS

THE EIGHTH INTERNATIONAL SYMPOSIUM
ON MULTIPLE-VALUED LOGIC

AD A 055763

AD NO. _____
DDC FILE COPY



Sheraton O'Hare Motor Hotel • Rosemont, Illinois

1978



78CH1366-4C

This document has been approved
for public release and sale; its
distribution is unlimited.

78 06 26 057

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM																
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER																
4. TITLE (and Subtitle) PROCEEDINGS OF THE 8TH INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC (8th).		5. TYPE OF REPORT & PERIOD COVERED Final rept.																
7. AUTHOR(s) Anthony S. Wojcik, Robert Swartwout		6. PERFORMING ORG. REPORT NUMBER																
9. PERFORMING ORGANIZATION NAME AND ADDRESS Anthony S. Wojcik, Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616		8. CONTRACT OR GRANT NUMBER(s) N00014-78-G-0019																
11. CONTROLLING OFFICE NAME AND ADDRESS Mathematical and Information Sciences Division Office of Naval Research		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 048-648																
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE May 1978																
		13. NUMBER OF PAGES 298																
		15. SECURITY CLASS. of this report 12/343p																
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE																
16. DISTRIBUTION STATEMENT (of this Report) Distribution of this document is unlimited.																		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)																		
18. SUPPLEMENTARY NOTES		<table border="1"> <tr> <td colspan="2">ACCESSION for</td> </tr> <tr> <td>DTIC</td> <td>White Section <input checked="" type="checkbox"/></td> </tr> <tr> <td>DDI</td> <td>Buff Section <input type="checkbox"/></td> </tr> <tr> <td colspan="2">UNANNOUNCED <input type="checkbox"/></td> </tr> <tr> <td colspan="2">JUSTIFICATION</td> </tr> <tr> <td colspan="2">BY</td> </tr> <tr> <td colspan="2">DISTRIBUTION STATEMENT CODES</td> </tr> <tr> <td colspan="2">Dist. to: ATAIL, ADG, or SPECIAL</td> </tr> </table>	ACCESSION for		DTIC	White Section <input checked="" type="checkbox"/>	DDI	Buff Section <input type="checkbox"/>	UNANNOUNCED <input type="checkbox"/>		JUSTIFICATION		BY		DISTRIBUTION STATEMENT CODES		Dist. to: ATAIL, ADG, or SPECIAL	
ACCESSION for																		
DTIC	White Section <input checked="" type="checkbox"/>																	
DDI	Buff Section <input type="checkbox"/>																	
UNANNOUNCED <input type="checkbox"/>																		
JUSTIFICATION																		
BY																		
DISTRIBUTION STATEMENT CODES																		
Dist. to: ATAIL, ADG, or SPECIAL																		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Multiple-Valued Logic																		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) PROCEEDINGS OF THE EIGHTH INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC																		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410 731



PROCEEDINGS

THE EIGHTH INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC



Sheraton O'Hare Motor Hotel • Rosemont, Illinois

1978

This document has been approved
for public release and sale; its
distribution is unlimited.

Additional copies may be ordered from:

IEEE Computer Society Publications Office
5855 Naples Plaza, Suite 301, Long Beach, CA 90803

acm Association for Computing Machinery
P.O. Box 12105, Church St. Station, New York, NY 10249

IEEE Service Center
445 Hoes Lane, Piscataway, NJ 08865



78 06 26 057

78CH1366-4C

TABLE OF CONTENTS

"A tri-State Logic Family"	1
S. C. Crist, Clarkson College	
"TTL Circuits for a Four Valued Bus"	7
D. Etienne, University of Paris	
"Logic Design of Multi-Valued I^2L Logic Circuits"	14
E. J. McCluskey, Stanford University	
"Some I^2L Circuits for Multiple-Valued Logic"	23
J. H. Pugsley and C. B. Silio, Jr., University of Maryland	
"On the Synthesis of Multi-valued Circuits Using Principally Binary Elements" . .	32
J. L. Huertas, J. I. Acha and L. Macias, Universidad de Sevilla	
"A Suggested Approach to Computer Arithmetic for Designers of Multi-Valued Logic Processors"	33
D. Akins, University of Michigan	
"Special Purpose Ternary Computer for Digital Filtering"	47
T. Higuchi and H. Hoshi, Tohoku University	
"Design and Implementation of a Non-binary Code for Byte-organized Memory with Binary and Quarternary Logics"	55
T. T. Dao, Sincetics Corp.	
"An Application of Multiple-valued Logic to a Design of Programmable Logic Arrays .	65
T. Sasao, Osaka University	
"A Simultaneous Analog-Ternary Converter"	73
H. A. H. Abdul-Karim and N. E. Berbat, University of Baghdad	
"Two Typical Representation Theorems for Symmetrical Heyting Algebras of Order n" .	76
L. Iturrioz, Université Claude-Bernard, Lyon	
"DeMorgan Algebras - Completeness and Recursion"	82
L. H. Kauffman, University of Illinois at Chicago	
"On the Compactification and Enumeration of Distinct Fuzzy Switching Functions" . .	87
A. Kandel, New Mexico Institute of Mining and Technology	
"Fuzzy Four-Valued Logic for Inconsistency and Uncertainty"	91
H. Stickel, University of Arizona	
"Four-Valued Threshold Logic Full Adder Circuit Implementations"	95
K. W. Current and D. A. Now, University of California at Davis	
"The Modular Complexity of a Tree Structured Higher Radix Multiplier"	101
J. Armstrong, VPI and State University	
"Addition in Signed Digit Number Systems"	104
M. Davio and J. P. Deschamps, MBL Research Laboratory, Brussels	
"A Simultaneous Radix 4, I^2L Multiplier Mechanized via Repeated Additions	114
A. D. Singh and J. R. Armstrong, VPI and State University	
"The Arithmetic Properties of Certain Number Systems"	122
M. H. Steward, California State University at Fullerton	
"Digital Calculus"	128
S. C. Lee and Y. M. Ajabnoor, University of Oklahoma	

"Completeness Problems for Switching Circuits Constructed from Delayed Gates" . . .	142
C. Reischer and L. Martin, Université de Quebec and I. C. Rosenberg, Université de Montreal	
"Complex Spectral Logic"	149
C. Moraga, Universität Dortmund	
"Use of an Infinite-valued Propositional Calculus in a Document Retrieval System" . .	157
J. R. Miller, Naval Research Laboratory	
"Mathematical Properties of Boolean Transformations"	163
Y. Levendel and M. A. Breuer, University of Southern California	
"Optimization of Multi-valued Decision Algorithms"	171
A. Thayse, M. Davio and J. P. Deschamps, IMBLE Research Laboratory, Brussels	
"Parallel and Serial Decompositions of Multi-valued Sequential Machines"	179
T. C. Yang and A. S. Wojcik, Illinois Institute of Technology	
"Applications of Multi-valued Logic in LSI Digital Signal Processing Circuits" . . .	187
K. W. Current and D. A. How, University of California at Davis	
"Iterative Realizations of Multi-valued Logic Systems"	188
V. P. Srin, Tennessee Technological University	
"Determination of the Fittest Number of Truth-values and Canonical Forms of Logical Functions for a Many-valued Axiom Set by a Computer"	195
M. Coto, S. Kao and T. Ninomiya, Meiju University	
"Vector Representation of Switching and Three-valued Functions"	202
Y. Levendel and M. A. Breuer, University of Southern California	
"Decomposition of Multiple-valued Logic Functions"	208
J. Fricke, Ruhr-Universität, Bochum	
"Theory and Design of Multi-valued Memory Elements"	213
J. L. Huertas, J. I. Acha and G. Sanchez-Gomez, Universidad de Sevilla	
"A Ternary J K Memory Element"	221
M. A. H. Abdul-Karim, University of Baghdad	
"A Behavioral Model and Triggering Modes for MVL-R-Flops"	226
M. S. Wills, USAF Systems Command	
"Generalized Finite Post Algebras"	235
J. C. Muzio, University of Manitoba and T. C. Wesselkamper, VPI and State University	
"Prefilters Over an Arbitrary Boolean Algebra"	242
I. H. Sack, Kean College of New Jersey	
"Automated Generation of Models and Counter-examples and Its Application to Open Questions in Ternary Boolean Algebra"	251
S. Winker and L. Mos, Aronne National Laboratory	
"A Summary of Investigations into Three and Four Valued Logics"	257
G. Epstein, Indiana University	
"A Proposed Interpretation of Lukasiewics' Four-valued System of Modal Logic"	258
S. J. Krolkoski, University of Illinois	
"A New Aspect of Some Post Algebras"	259
S. Perrine, Metz, France	

"Analysis of Computing Protection Structures by Means of Multi-valued Logic Systems".	260
L. J. Kohout, University of London	
"The B-Ternary Logic and Its Application to the Detection of Hazards in Combinational Switching Circuits"	269
M. Mukaidono, Meiji University	
"Possibility Theory: as a Means for Modeling Computer Security and Protection" . . .	276
D. C. Pine, Western Illinois University	
"Fault Secure Multiple-valued Logic Networks"	287
J. E. Smith, University of Wisconsin-Madison and J. Dussault, Western Electric Company	

P

APPROVED FOR	
WHS	W.D. Section <input checked="" type="checkbox"/>
DOC	B.H. Section <input type="checkbox"/>
UNCLASSIFIED	
S I C I T I O N	
DISSEMINATION/AVAILABILITY CODES	
A	SPECIAL

A TRI-STATE LOGIC FAMILY

Stephen C. Crist
Clarkson College of Technology
Potsdam, New York

ABSTRACT

A philosophy for implementation of three-value logic is introduced utilizing the tri-state concept found in conventional T^2L buffers. By utilizing the high impedance state as a distinct logic input as well as output, a three-valued logic family can be constructed. Circuits that realize one functionally complete set of three-valued functions are described. The circuits are basically T^2L in nature, and retain many of the desirable properties of that family. One supply is required, and no level conversion circuitry is necessary to interface between the tri-state circuits and conventional binary T^2L .

I. INTRODUCTION

A conventional T^2L totem-pole output stage is depicted in Figure #1. In the currently available tri-state buffers, there are three states that the output can take on: Q1 on and Q2 off (Logic 1), Q1 off, and Q2 on (Logic 0), and both transistors off. The last state presents a very high impedance.

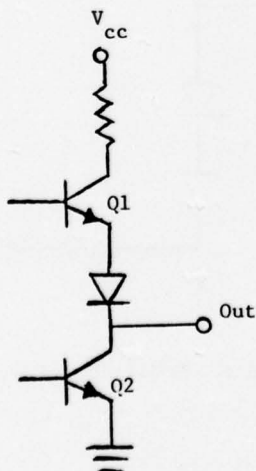


Figure 1. T^2L output stage

A conventional T^2L input stage, as shown in Figure #2, reacts to a high impedance input as it does to a Logic 1 input, because R_1 is normally chosen such that sufficient base current is available to Q2 through R_1 without external current.

A typical RTL input stage is shown in Figure #3. If, as in T^2L , the two conventional logic outputs are current drive (logic 1) and current sink (logic 0), then a high impedance input will behave as a logic 0 input because it can provide no base current to the input transistor.

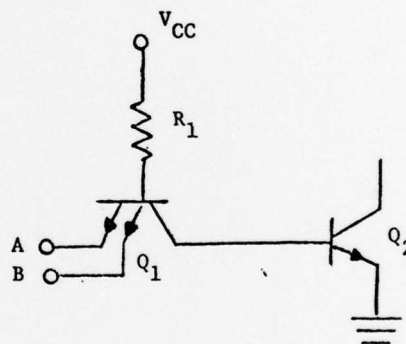


Figure 2. TTL Input Stage

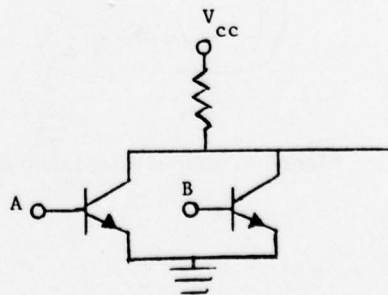


Figure 3. RTL input stage

Neither input described can discriminate the open state from both logic 0 and 1, but a combination of the two input stages can because they identify the high impedance as a different value. This is the general philosophy of the circuits described in this paper. The high impedance state is defined as logic 2.

For any circuit so constructed, the input would appear as shown in Figure #4. This configuration poses no loading problem when the input is logic 0 or 1, but if the input is an open circuit, a problem exists. Because logic 2 is to appear as a logic 1 to Q1, Q1 should conduct in the reverse direction. Logic 2 is to appear as a logic 0 to Q2, thus no base current should flow in Q2. When the two input stages are connected together, however, current can flow in the path shown. The solution is shown in Figure #5. In standard T²L, as well as the circuits later described herein, a voltage of 3 diode drops on the base of Q1 will cause the collector-base junction to become forward biased and thus Q1 will conduct in the reverse direction. This voltage on the base of Q1 is not sufficient for current to flow through the emitter of Q1 to the base of Q2, thus Q2 will remain off as desired.

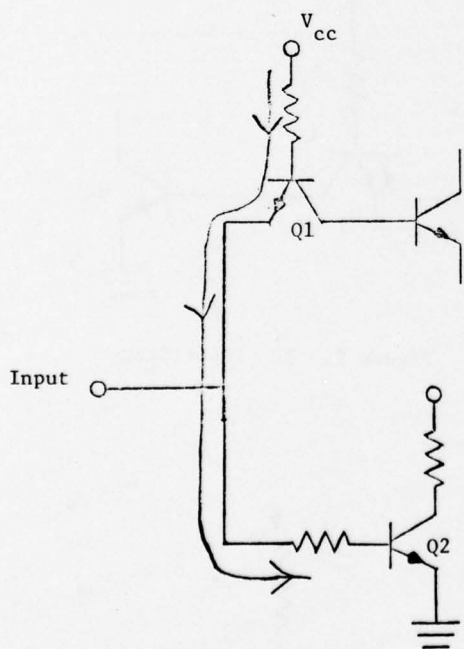


Figure 4. Tri-State input stage

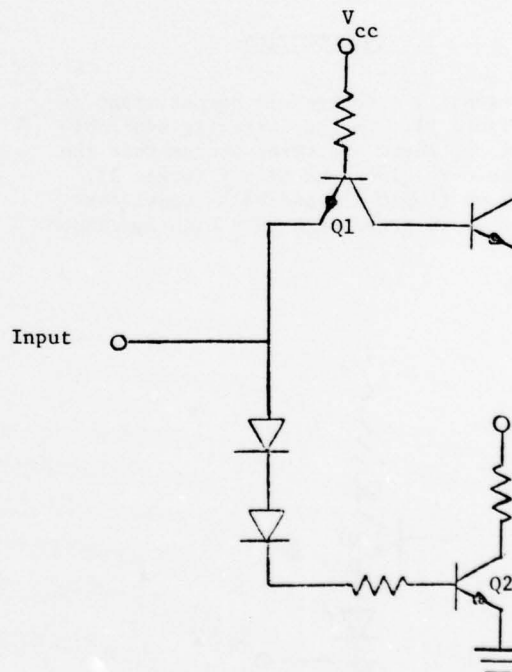


Figure 5. Modified tri-state input stage

II. A POSSIBLE IMPLEMENTATION

The following algebra was chosen for implementation⁸.

Cycle: $\vec{F}^B = A + B \text{ mod } 3. \quad (B = 1 \text{ or } 2)$

Complement: $F = \bar{A} = 2 - A$

Max: $F = A + B = \text{maximum value of } A \text{ and } B$

Min: $F = A \cdot B = \text{minimum value of } A \text{ and } B$

Because the minimum function can be generated by the maximum and complement functions, a functionally complete set of operations need not contain minimum. The cycle functions are constructed with B a constant 1 or 2. The cycle 2 function is also redundant because $\vec{A}^2 = (\vec{A}^1)^1$, but is included.

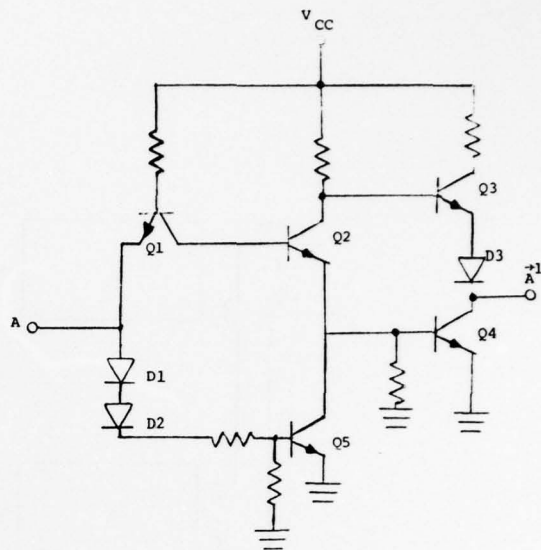
Figure #6 shows a cycle 1 gate. Q5 turns on only when A is logic 1. If A is not equal to logic 1, then Q5 is an open circuit and does not affect the circuit. The remainder of the circuit is a \overline{A}^2 inverter. Thus, a 0 input yields a 1 output, and a 2 input behaves as a 1 input and gives a zero output. When the input is logic 1, Q2 turns on, attempting to drive the totem-pole into the zero state. However Q5 will prevent this by turning on and keeping the base of Q4 below cut-in, thus forcing \overline{A}^1 to logic 2.

The general philosophy of the \overline{A}^1 circuit is used in all the other circuits described herein. The circuits can be divided into two parts, as shown in Figure #7. If circuit B is ignored, circuit A will always produce a logic 0 or 1 output. Circuit A is designed so that the output is correct when it is supposed to be a 0 or 1 assuming QB is off. Circuit B turns QB off if the output is to be 0 or 1. If the output is to be a logic 2, the collector of QB is connected to the base of either QA₂, or QA₃, depending on which transistor is turned on by circuit A. Since QB is saturated when it is on, it can force either output transistor to turn off.

This scheme has been tested on the \overline{A}^1 and A + B circuit, and was functional. It is noted, however, that if the collector of QB is connected to QA₃, the base-emitter junction of QA₃ is not reverse biased, although the forward V_{BE} is small.

This can be alleviated by the configuration of Figure 8. (This applies only to circuits that must turn off QA₃.) QB turns off QA₁, insuring QA₃ will be off.

QB₂ is necessary because otherwise QA₂ would turn on.



A	\overline{A}^1
0	1
1	2
2	0

Figure 6. \overline{A}^1

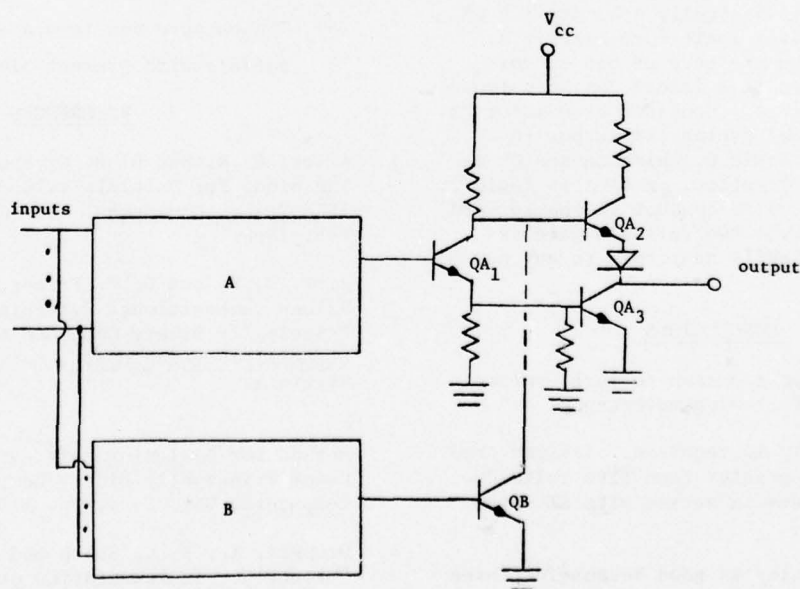


Figure 7. General tri-state configuration

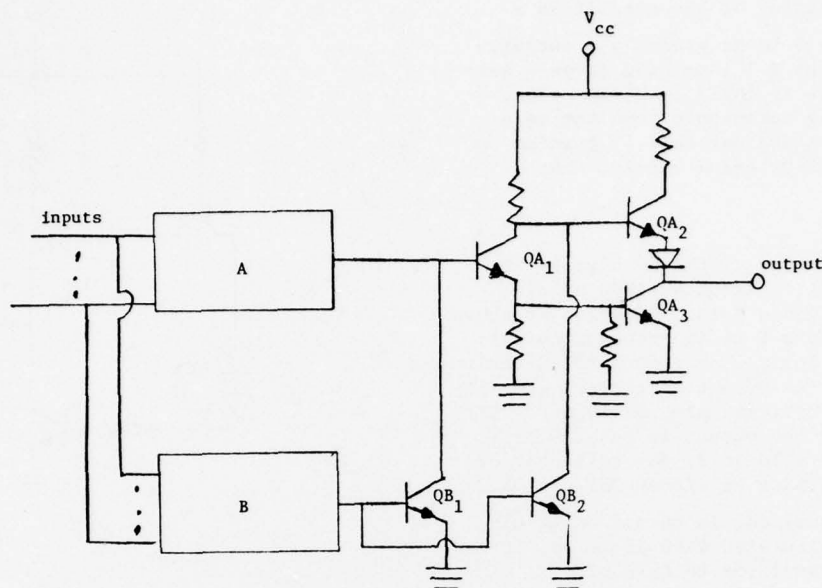


Figure 8. Modified tri-state configuration

The \bar{A}^1 , \bar{A}^2 , and \bar{A} circuits are shown in Figures 9, 10 and 11 respectively, with the modification suggested in Figure #8 made to the \bar{A}^1 gate. The same general philosophy is applied to the maximum gate of Figure #12. Circuit A is basically a binary T^2L OR gate. Logic 2 appears as a logic 1 to circuit A. Circuit B must detect the presence of one or more logic 2 inputs, thus circuit B itself has both types of inputs for each variable. Consider transistor Q9. It will turn Q10 on, thus forcing the output to Logic 2, unless a) A is logic 0, which causes Q9 to conduct in the forward direction, or b) A is logic 1, in which case Q8 forces Q9 to conduct in the forward direction. Although only a two variable gate is shown, the circuit is readily adaptable to any number of inputs.

III. CONCLUSIONS

It is felt that this approach to three-valued logic circuits has the following advantages:

- 1) One power supply is required. (It may need to be slightly greater than five volts due to the two diodes in series with RTL-type inputs.)
- 2) The noise immunity is good because of three distinct states of current at the output.
- 3) There is little dependence on resistor values or active device parameters for correct

operation of the circuits.

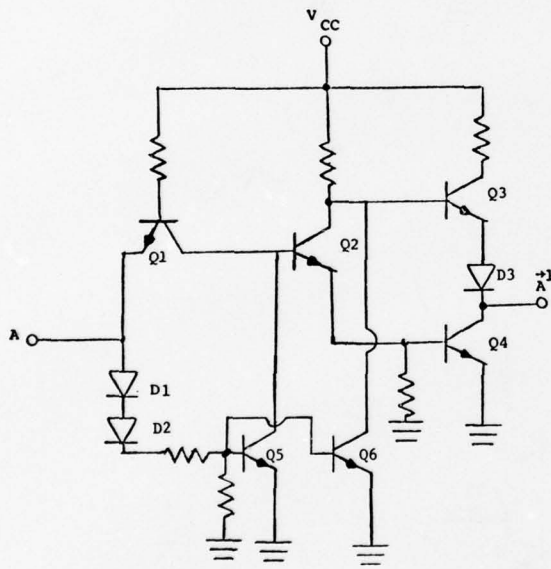
- 4) The circuits can be fabricated using existing T^2L technology.
- 5) The outputs and inputs are electrically compatible with present binary T^2L circuits.

REFERENCES

1. Allen, C. M. and D. D. Givone, "A Minimization Technique for Multiple-Valued Logic Synthesis," IEEE Trans. Computers, Vol. C-17, pp. 182-189, Feb. 1968.
2. Birk, J. E. and D. E. Farmer, "Design of Multi-Valued Combinational Switching Circuits Using Principally Binary Components," 1974 Int. Symp. on Multiple Value Logic, Morgantown, West Virginia.
3. _____, "An Algebraic Method for Designing Multi-Valued Logic Circuits Using Principally Binary Components," IEEE Trans. Computers, Vol. C-24, pp. 1101-1104, Nov. 1975.
4. Druzeta, A., K. C. Smith and Z. E. Vranesic, "Electronic Implementation of Multi-Valued Logic Networks," 1974 Int. Symp. on Multiple Value Logic, Morgantown, West Virginia.
5. Etienne, D. and M. Israel, "Implementation of

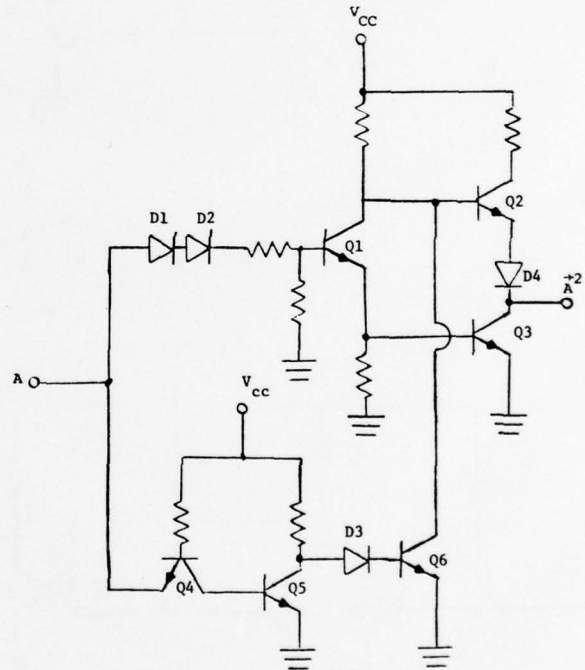
Ternary Circuits with Binary Integrated Circuits,"
1977 Int. Symp. on Multiple Value Logic, Charlotte,
N.C.

6. _____, "A New Concept for Ternary Logic Elements," 1974 Int. Symp. on Multiple Value Logic, Morgantown, West Virginia.
7. _____, "TTL Circuits for Multi-Valued Boolean Algebra," 1977 Int. Symp. on Multiple Value Logic, Charlotte, N.C.
8. Irving, T. A., H. T. Nagle and S. G. Shiva, "Flip-Flops for Multiple Valued Logic," IEEE Trans. Computers, Vol. C-25, pp. 237-240, March, 1976.
9. Jordon, F. B. and H. T. Mouftah, "Integrated Circuits for Ternary Logic," 1974 Int. Symp. on Multiple Valued Logic, Morgantown, West Virginia.
10. Mouftah, H. T., "A Study of Implementation of Three Valued Logic," 1976 Int. Symp. on Multiple Value Logic, Logan, Utah.



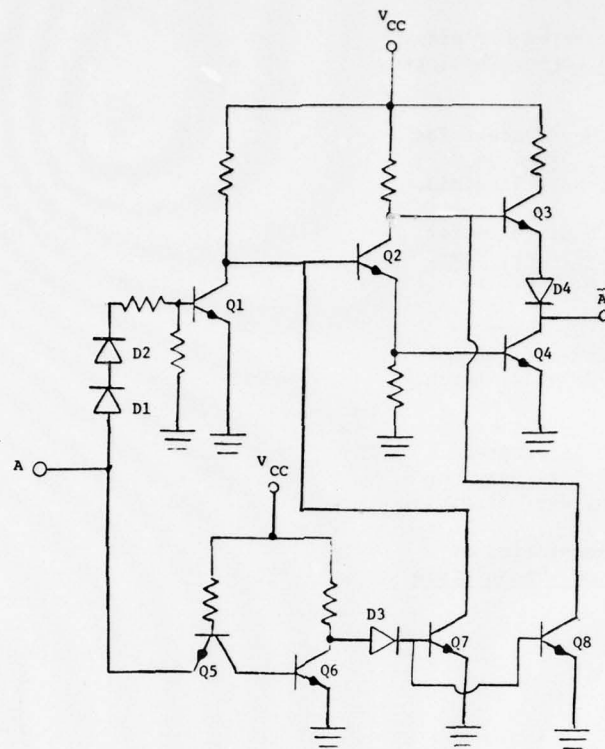
A	A ⁺¹
0	1
1	2
2	0

Figure 9. $A \rightarrow A^+1$



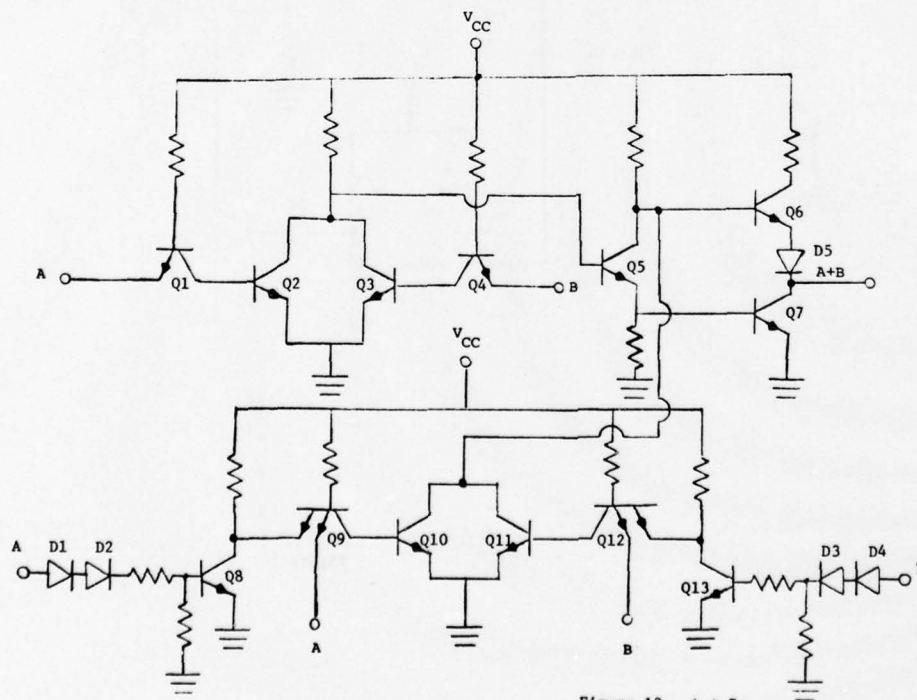
A	A ⁺²
0	2
1	0
2	1

Figure 10. $A \rightarrow A^+2$



A	\bar{A}
0	1
1	0

Figure 11. \bar{A}



A	B	A+B
0	0	0
0	1	1
0	2	2
1	0	1
1	1	1
1	2	2
2	0	2
2	1	2
2	2	2

Figure 12. $A + B$

TTL CIRCUITS FOR A 4-VALUED BUS
A way to reduce package and interconnections

Daniel ETIEMBLE

UNIVERSITY PARIS 6

Abstract

This paper presents voltage mode multi-valued circuits to define a 4-valued bus. Two different versions are presented : TTL circuits for a 4-valued open collector bus, and TTL circuits for a 4-valued + high impedance bus. Some dynamic characteristics are shown. With usual load of a bus, the supplementary delay is less than 55 ns.

I - INTRODUCTION

Integrated circuit technology has made great advances in the past fifteen years. "Relating monolithic processor advances to large scale integration processing progress and assuming election beam lithography to replace photomask lithography, 1970-77 growth in monolithic processor sophistication and complexity can be expected to continue as before and double each year (Moore-Noyce Rule)". [1]. One of the most important problems is the pin limitation of the integrated circuits. Complexity of a system is proportional to the number of chips and interconnections. The most commonly used method to reduce the number of pins is multiplexing. While in applications with relatively slow processor elements the time multiplex solution is preferred, in high speed applications the space multiplex may be better. Multi-valued circuits offer another solution. They allow each input pin to accept and each output pin to deliver more information.

Since the beginning of 1960'S, many multi-valued circuits have been presented [2]. In 1974, there appeared papers with 3-valued TTL circuits [3] and 3-valued COSMOS circuits [4]. In 1976, Signetics corporation presented multivalued current-mode circuits, using I²L technology [5]. These circuits can be used to implement threshold logic. As shown in reference [6], these circuits use multi-valued currents inside the chip. But noise margins considerations forbid the use of multi-valued current outside the chip. To reduce interconnections by reducing the number of pins, one need multi-valued voltage inputs and outputs.

In this paper, voltage mode multi-valued circuits are described to define a 4-valued bus.

II - A 4-VALUED BUS

Microprocessor devices, memories, etc, are 2-valued integrated circuits. It has been shown [7, 8] that 4-valued algebras are generalized boolean algebras. By embodying 2-valued circuits with an input decoder and an output encoder, it is easy to define 4-valued integrated circuits (Fig.1).

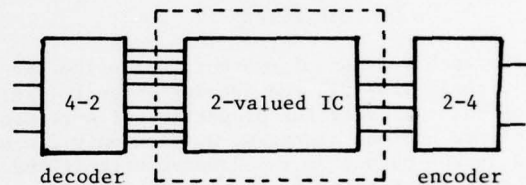


Fig. 1 4-valued IC.

The decoder and the encoder circuits must be integrated on the same chip as the 2-valued circuit. They use the same technology or a compatible technology as the 2-valued circuit. With processors using N-MOS technology, the decoder and encoder circuits must use N Channel MOS technology. TTL interface circuits are compatible with I²L or I³L technology.

The decoder and encoder circuits may be considered as interface circuits between the commonly used 2-valued integrated circuits and the 4-valued bus, but these interface circuits are integrated inside the chip. Outside the chip, the entire circuit looks like a 4-valued circuit. This method has some advantages and disadvantages.

Advantages :

- this method is the most efficient method to implement 4-valued circuits with technologies which are typically 2-valued ones [6].
- It allows us to use the usual 2-valued microprocessors, RAM, ROM, etc., with a slight modification, as 4-valued circuits.

Disadvantages :

- the speed of 4-valued circuits is lower than the speed of the 2-valued corresponding ones. We must add to the delay time of the 2-valued circuits the delay time of the two interface circuits. It would make no sense to apply this approach to SSI circuits, since the two conversions delay times would be greater than the processing time. But with LSI

circuits as microprocessors or memories, the time required for the 2 conversions is only a small part of the delay time of the 2-valued LSI circuits. The decrease in speed is compensated by the drastic decrease in the number of required pins.

We present here TTL circuits for a 4-valued bus. The load of the 4-valued bus is the same as that of the corresponding 2-valued bus. The TTL circuits are compatible with I²L technology. Two different versions are presented :

- 4-valued open collector bus
- 4-valued + high impedance bus.

We adopt the following notational conventions

- . $E_4 = \{0, 1, 2, 3\}$ Set of 4 values
- . $E_2 = \{0, 1\}$ Set of binary values
- . $a, b, c, x, y \in E_2$.
- . $z \in E_4$.
- . Let z^i be a 2 valued function such that

$$z^i = 1 \text{ when } z \geq i$$

$$= 0 \text{ otherwise}$$

For each version of the bus, we define the encoder circuit (Fig.2), the decoder circuit (Fig.3) and the coding. Using the properties of 4-valued generalized boolean algebras, we need only 2 binary values at the output of the decoder circuit and at the input of the encoder circuit (Fig. 4).

Coding determines the relations between

- a) x_i, y_i and a, b, c for the encoder circuit
- b) x_o, y_o and $\bar{z}^1, \bar{z}^2, \bar{z}^3$ for the decoder circuit.

There are 16 different ways of coding, but only 2 fundamental codes. The other ones correspond to permutations between x and y , x and \bar{x} , y and \bar{y} . For each version, we give only the two fundamental codes.

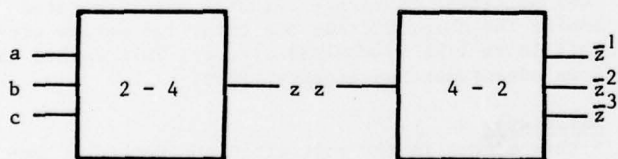


Fig. 2 Encoder circuit Fig. 3 Decoder circuit

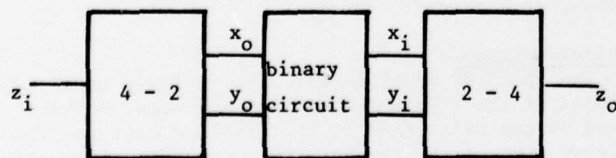


Fig. 4 Interface circuit

III - 4-VALUED OPEN COLLECTOR TTL CIRCUITS

III.1 - Encoder circuit

We use the TTL 7403 circuit (quadruple 2 input NAND gates with open collector outputs). The encoder circuit is shown in Fig.5. Table I gives the relation between the logical values and the operating voltage levels. We may use more diode drops to define the operating levels corresponding to logic values 1 and 2 to improve noise margins. This would need some modifications in the decoder circuit.

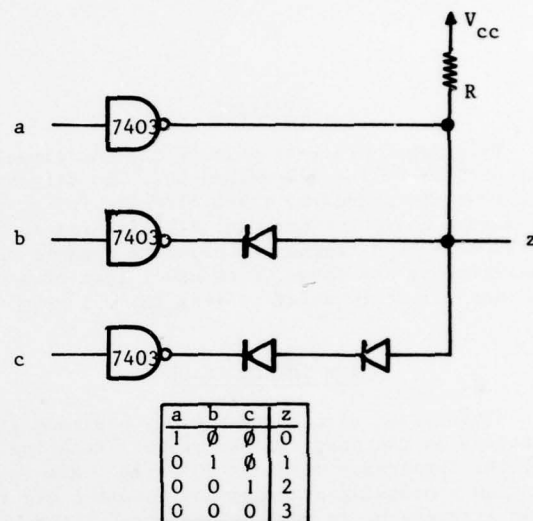


Fig. 5 : open collector encoder circuit

Logical values	Operating levels
0	$V_{ce \text{ sat}}$
1	$V_{ce \text{ sat}} + 1 \text{ diode drop}$
2	$V_{ce \text{ sat}} + 2 \text{ diodes drops}$
3	V_{cc}

Table I

III.2 - Decoder circuit

The decoder circuit implements the \bar{z}^1, \bar{z}^2 and \bar{z}^3 functions. With the input voltage levels defined by the encoder circuit, the decoder circuit is shown in Fig.6. It uses 3 7404 gates and 2 diodes drops. The output levels corresponding to \bar{z}^1, \bar{z}^2 and \bar{z}^3 are the usual level of TTL technology, except for the low level of \bar{z}^3 , which is one diode drop above the usual level. A slight modification in the integrated version of the \bar{z}^3 circuit would give the usual low level.

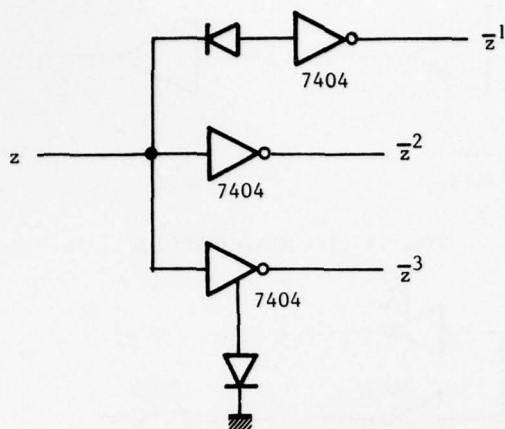


Fig. 6 Open collector decoder circuit

III.3 - Coding

The first fundamental code is given in Table II.

a	b	c	z	\bar{z}^1	\bar{z}^2	\bar{z}^3	x_i	y_i
1	0	0	0	1	1	1	1	1
0	1	0	1	0	1	1	1	0
0	0	1	2	0	0	1	0	1
0	0	0	3	0	0	0	0	0

Table II

By using the don't cares, we get the following relations.

$$\begin{aligned} a &= x_i \cdot y_i & x_0 &= \bar{z}^2 \\ b &= x_i & y_0 &= \bar{z}^1 + z^2 \cdot \bar{z}^3 = \bar{z}^3 + \bar{z}^2 \cdot z^1 \\ c &= y_i \end{aligned}$$

The corresponding implementation is given in Fig. 7

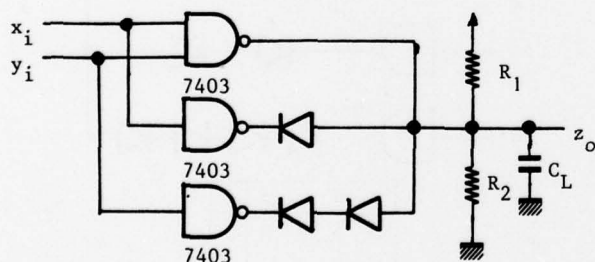


Fig. 7 First implementation corresponding to first code

As the operating levels corresponding to x_i and y_i inputs are the output levels of 2-valued circuits inside the chip, we may simplify the scheme of Fig. 7 by using complemented values of x_i and y_i .

Then, the relations become

$$\begin{aligned} a &= \bar{x}_i \cdot \bar{y}_i = \overline{x_i + y_i} & x_0 &= z^2 \\ b &= \bar{x}_i & y_0 &= z^3 + \bar{z}^2 \cdot z^1 \\ c &= \bar{y}_i \end{aligned}$$

The corresponding implementation is given in Fig. 8

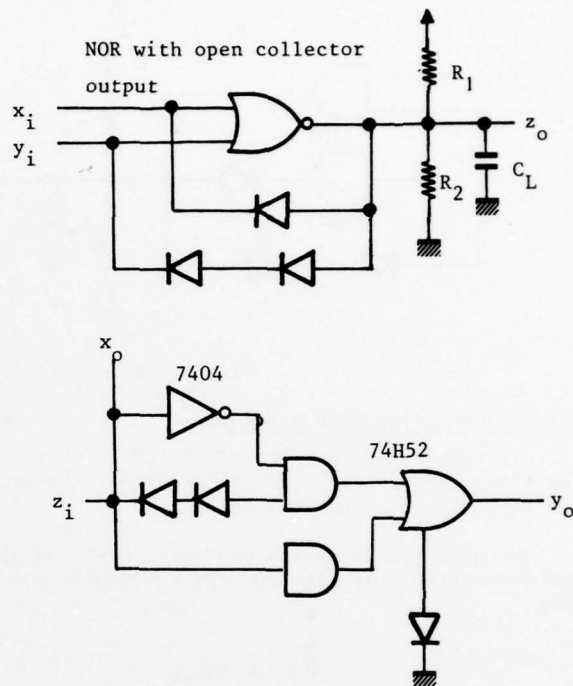
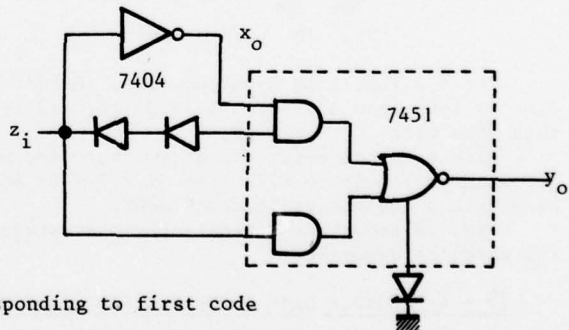


Fig. 8 Second implementation corresponding to first code.



The second fundamental code (Gray code) is given in Table III.

a	b	c	z	\bar{z}^1	\bar{z}^2	\bar{z}^3	x_i	y_i
1	0	0	0	1	1	1	1	1
0	1	0	1	0	1	1	1	0
0	0	1	2	0	0	1	0	0
0	0	0	3	0	0	0	0	1

Table III.

The corresponding relations are :

$$\begin{aligned} a &= x_i \cdot y_i \\ b &= x_i \\ c &= \bar{y}_i \end{aligned} \quad \begin{aligned} x_o &= \frac{-2}{z^2} \\ y_o &= \frac{-3}{z^3 \cdot z^1} \end{aligned}$$

The resultant implementation is given Fig. 9

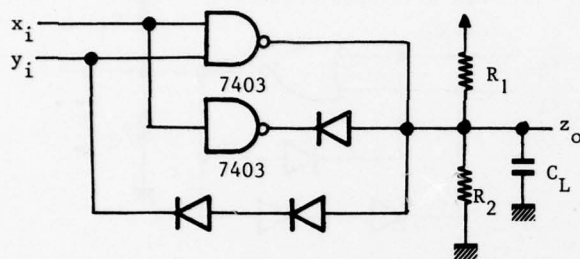


Fig. 9 Implementation corresponding to Gray code

III.4 - Operating Characteristics

To each coding corresponds 3 levels of gate. The first one has two dynamical hazards. The second one has only one dynamical hazard.

We give the operating characteristics using the load circuit of Fig. 10, and the second coding.

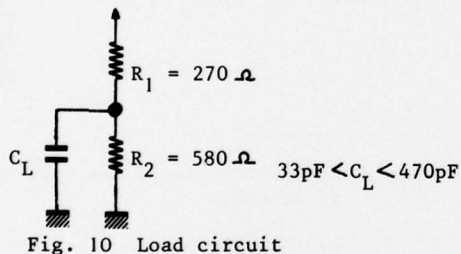


Fig. 10 Load circuit

For a 2 Foot long 4-valued wire, the delay time is less than 37ns with $C_L = 33$ pF, and less than 55ns with $C_L = 150$ pF.

With open collector TTL circuits, noise margin is good enough to allow use of a 2 foot long wire with a capacitive load of 150pF.

Fig. 18 shows the dynamic characteristics of the circuits described.

IV - 4-VALUED + HIGH IMPEDANCE TTL CIRCUITS

This version is an extension of the tri-state circuits of the binary case. We use an active pull up circuit which is controlled by a special input.

IV. 1- Encoder circuit.

We use 74125 or 74126 Tri state Buffers.

Fig. 11 shows the circuits and their logical function.

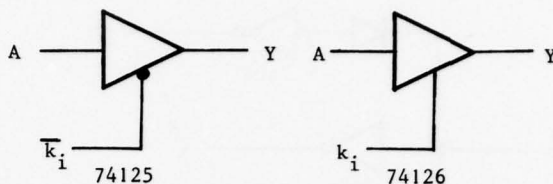
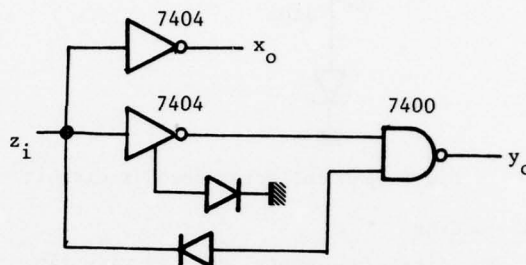


Fig. 11 Tri State Buffers



A	\bar{k}_i	Y
0	0	0
1	0	1
0	1	Z

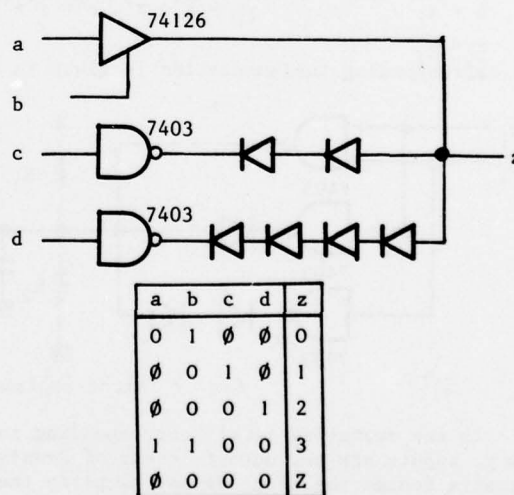
74125

A	k_i	Y
0	0	Z
0	1	0
1	1	1

74126

Fig. 11 : Tri State Buffers

The encoder circuit is shown in Fig. 12



a	b	c	d	z
0	1	0	0	0
0	0	1	0	1
0	0	0	1	2
1	1	0	0	3
0	0	0	0	Z

Fig. 12 : Encoder circuit

Table IV give the relation between logical values (0, 1, 2, 3) and the operating levels.

Logical values	operating levels
0	$V_{ce\ sat}$
1	$V_{ce\ sat} + 2\text{ diodes drops}$
2	$V_{ce\ sat} + 4\text{ diodes drops}$
3	TTL high level
Z	High impedance state

Table IV

We use 2 diodes drops and 4 diodes drops for noise margins considerations.

IV. 2 - Decoder circuit.

The \bar{z}^1 , \bar{z}^2 functions are implemented by the circuits shown in Fig. 13. An integrated version of \bar{z}^3 circuit is shown in Fig. 14.

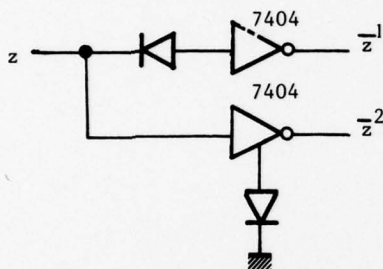


Fig. 13 : \bar{z}^1 and \bar{z}^2 circuits

IV. 3 - Coding

We use only the second fundamental coding (Table V).

a	b	c	d	z	\bar{z}^1	\bar{z}^2	\bar{z}^3	x_i	y_i	k_i
0	1	0	0	0	1	1	1	0	1	1
0	0	1	0	1	0	1	1	0	0	1
0	0	0	1	2	0	0	1	1	0	1
1	1	0	0	3	0	0	0	1	1	1
0	0	0	0	Z	?	?	?	0	0	0

Table V

k_i is the control input of the high impedance state. When $k_i = 0$, the output z is in the high impedance state. It would make no sense to decode this state. The ? means that the outputs of the decoder circuits do not deliver information.

The corresponding relations are :

$$\begin{aligned}
 a &= x_i \cdot k_i & x_o &= \frac{\bar{z}^2}{\bar{z}^3 \cdot z^1} \\
 b &= y_i \cdot k_i & y_o &= \frac{\bar{z}^3}{\bar{z}^3 \cdot z^1} \\
 c &= \bar{x}_i \cdot k_i \\
 d &= \bar{y}_i \cdot k_i
 \end{aligned}$$

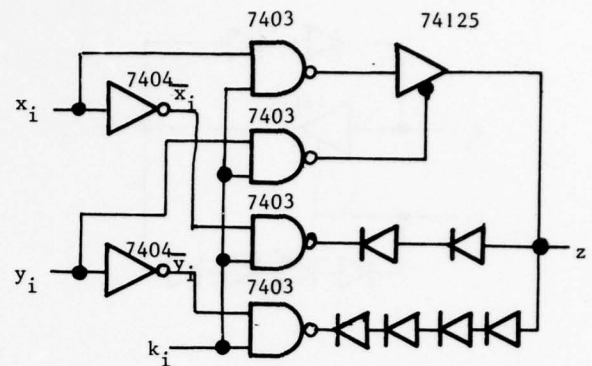


Fig. 15 Implementation of second coding

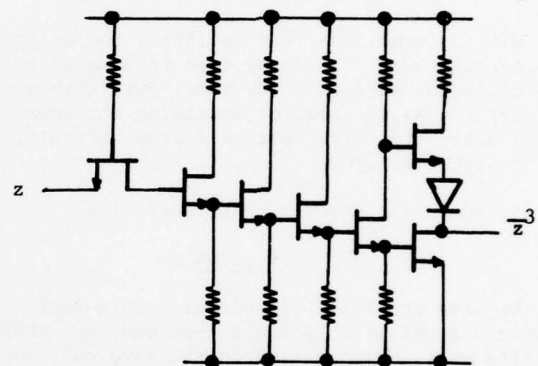


Fig. 14 : Integrated version of \bar{z}^3 circuit.

A slight modification in the 74126 circuit would give the scheme shown in Fig. 16. It only means the use of multi emitter transistors for the A inputs and for the control inputs.

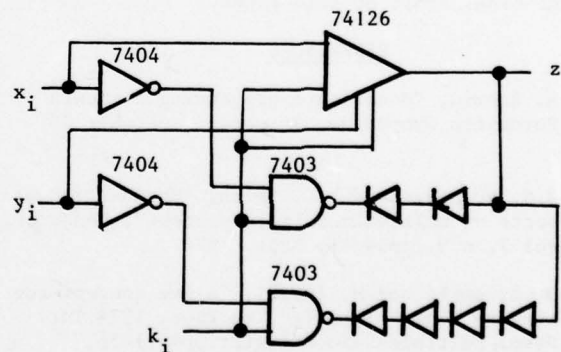


Fig. 16 : Modified Implementation.

IV. 4 - Operating characteristics

The results shown correspond to the 4-valued circuit shown in Fig. 17. The gates corresponding to the High impedance control have been deleted. This scheme allows us to examine the switching characteristics of a 4-valued circuit with active pull up output.

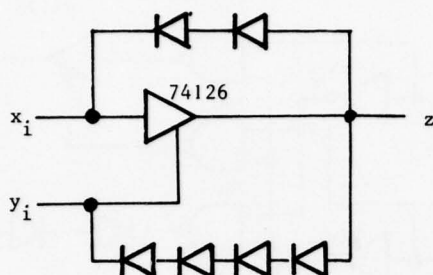


Fig. 17 : 4-valued circuit with active pull up output

With a capacitive load of 150pf, the delay time is less than 50ns. The delay time is similar to the open collector version delay time. But, with active pull up, we may use smaller switching currents. (For a 55ns delay time with open collector circuits, the load resistor is 270 Ω).

Fig. 19 shows the dynamic characteristics.

V - CONCLUSION

We have presented TTL circuits as interface between a 2-valued chip and a 4-valued bus. These circuits must be integrated on the same chip as the 2-valued one. Two different versions have been presented. It has been shown that with usual load of a bus, the supplementary delay is less than 55 ns.

ACKNOWLEDGMENT

The author wishes to thank Prof Z. Vranesic (University of Toronto) for his helpful comments for the final draft of this paper.

REFERENCES

- 1 H. Schmid, "Monolithic processing elements", Euromicro Symposium, Amsterdam, October 77,
- 2 Z.G. Vranesic and K.C. Smith, "Engineering aspects of multivalued logic systems", Computer, vol 7, n°9, pp34-41, Sept. 1974
- 3 D. Etiemble and M. Israël, "A new concept for ternary logic elements", in Proc. 1974 Int. Symp. Multiple-Valued Logic, pp437-455, May 1974.
- 4 H. Mouftah and I Jordan, " Integrated circuits For ternary logic", in Proc. 1974 Int. Symp. Multiple-valued Logic, pp 285 - 302, May 1974.
- 5 T.T. Dao, E. J. Mc Cluskey and L.K. Russel, "Multivalued Integrated Injection Logic", IEEE Trans. Comput, Vol. C 26, pp 1233 - 1241, Dec. 1977.
- 6 D. Etiemble, "Etude critique des circuits mul-

tivalués pour le traitement de l'information". Thèse d'Etat à paraître.

7 S.C. Lee and Y.K. Zvi, "A generalized boolean algebra and its application to logic design", in Proc. 1975 Int. Symp. Multiple-Valued Logic, pp88-98, May 1975.

8 J.Dussault, G. Metze and M. Krieger, "A multi-valued switching algebra with boolean properties", in Proc. 1976 Int. Symp. Multiple-valued Logic, pp 68-73, May 76.

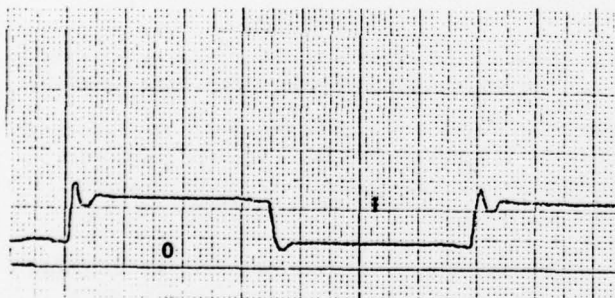
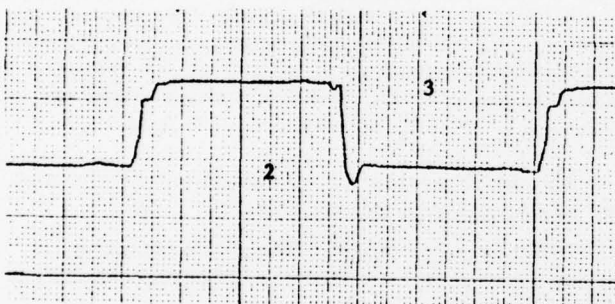
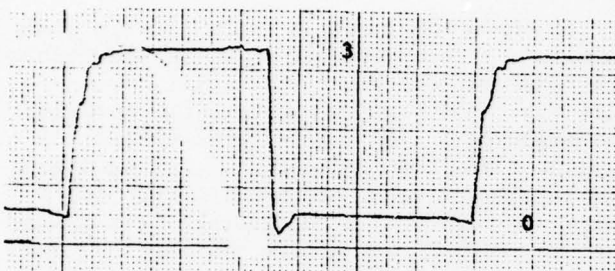
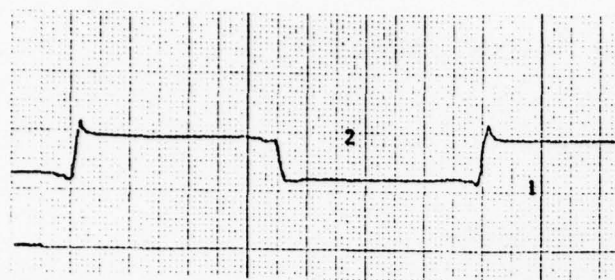


Fig. 18 Open collector TTL circuits characteristics.

1V/cm
100 ns/cm

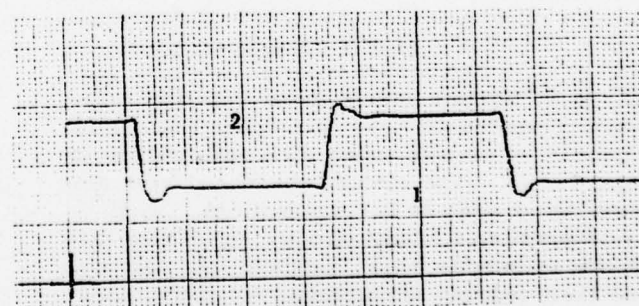
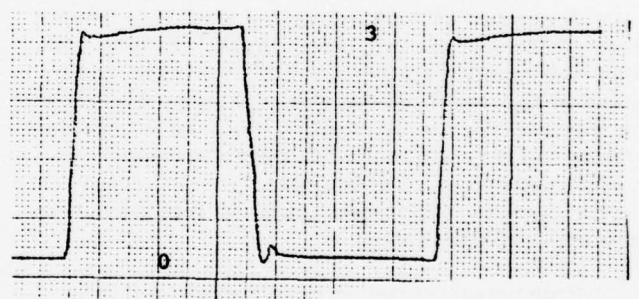
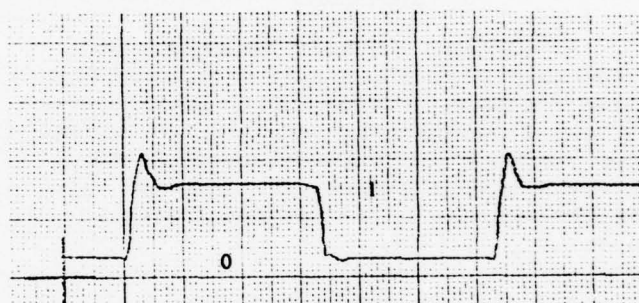
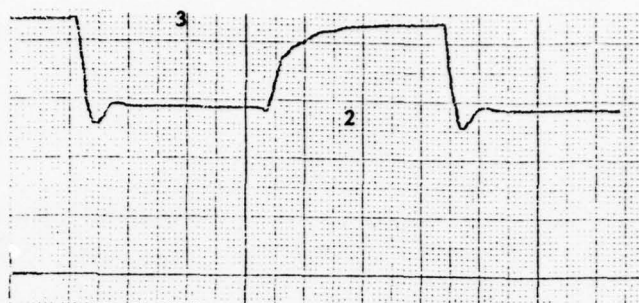


Fig. 19 4-valued TTL circuits with active pull up output characteristics.

1V/cm
100 ns/cm

LOGIC DESIGN OF MULTI-VALUED I^2L LOGIC CIRCUITS

E. J. McCluskey*

Consultant, Signetics Corporation
Sunnyvale, California 94086

ABSTRACT

An algebraic system for designing multi-valued (four-level) I^2L circuits is presented here. It was necessary to develop this system because the use of existing multi-valued logic formalisms does not result in efficient I^2L circuits. The close relationship between the algebra and the integrated circuits is stressed throughout the paper.

INTRODUCTION

A number of techniques for the logic design or minimization of multilevel digital circuits have been previously published.^{1,2,3,4} These papers differ in the basic connectives chosen, but have the common feature of starting with an algebraic system. They either do not consider circuit implementation or exhibit only "proof of feasibility" circuits rather than circuits intended for practical implementation.

This paper presents an algebraic system which was developed from a family of four-level I^2L circuits which have actually been constructed using a standard I^2L process.^{5,6} This circuit family is called Multi-valued I^2L or MVI 2L . Although the discussion is given in terms of four-level circuits (since these were the ones actually fabricated), the techniques are easily extended to circuits using different numbers of logic levels. The circuits discussed here are intended for MSI or LSI applications. Their advantages over binary circuits arise from their ability to provide more "function" per unit area and their reduced requirements for interconnections due to the fact that fewer four-level signals than two-level signals are necessary to represent the same information.

In two-valued or binary digital circuits each signal can take on one out of two possible values and represents one bit of information. The precise numerical values assumed by the variables are not important for logic design and thus the symbols 0 and 1 are typically used to represent the two "logic" values assigned to binary variables. The circuits to be described here operate with four-valued or quaternary signals. The symbols 0, 1, 2 and 3 will be used to represent the logic values of the signals. In the actual implementations the four signal values are represented by four different values of current (actual current magnitudes are controlled by the speed requirements of the circuit). Typical values might be $<0, 10 \mu A, 20 \mu A, 30 \mu A>$.

The logic system to be described here makes use of three types of operators:

- (1) A set of two-valued (binary) functions of a single (quaternary) variable. These functions are called literals.
- (2) A four-valued function of a single (quaternary) variable called the complement.
- (3) Three connectives--PLUS, MAX, and INHIBIT--for combining two or more quaternary variables.

These operators were selected because their use leads to efficient circuits. In fact they were chosen by studying already designed circuits in order to identify convenient building blocks.

The symbology used is summarized in Table I which should be helpful as a reference when studying the design description.

LITERALS

Literals are binary functions of a single quaternary variable. ** The most primitive literals

**The strong U Threshold literals are called "step-up" functions and the strong D Threshold literals are called "step-down" functions by Ying.⁴ They are credited by Ying to Muhldorf. The delta

*Digital Systems Laboratory, Stanford University, Stanford, Ca. 94305

are the weak threshold literals, $u_t(X)$ and $d_t(X)$, defined by

$$\begin{aligned} u_t(X) &> 0 \text{ iff } X \geq t \\ u_t(X) &= 0 \text{ iff } X < t \end{aligned} \quad (1)$$

and

$$\begin{aligned} d_t(X) &> 0 \text{ iff } X \leq t \\ d_t(X) &= 0 \text{ iff } X > t \end{aligned} \quad (2)$$

where X is a quad variable taking on only values 0, 1, 2 or 3; and t is a rational number between 0 and 3. For example $u_2(X) = \langle 0, 0, +, + \rangle$ where the bracket notation shows the result when X takes on the successive values 0, 1, 2, 3 and the + indicates any non-zero value. Similarly, $d_2(X) = \langle +, +, +, 0 \rangle$ where only values of $X \leq 2$ (down from 2) will be non-zero.

Related to the weak threshold literals are the strong threshold literals, $U_t(X)$ and $D_t(X)$, which are defined by

$$U_t(X) = 1 \text{ iff } X \geq t \quad (a) \quad (3)$$

$$U_t(X) = 0 \text{ iff } X < t \quad (b)$$

and

$$D_t(X) = 1 \text{ iff } X \leq t \quad (a) \quad (4)$$

$$D_t(X) = 0 \text{ iff } X > t \quad (b)$$

The two classes of threshold literals differ in that the weak literals require only a value greater than zero* when the appropriate condition is satisfied, while the strong functions require a value of 1 when the appropriate condition occurs. For example, $U_2(X) = \langle 0, 0, 1, 1 \rangle$ and $D_2(X) = \langle 1, 1, 1, 0 \rangle$.

A circuit for realizing the threshold literals is shown in Fig. 1. Circuits will be presented here to illustrate the complexity involved, but the functioning of the circuits will not be discussed. Readers interested in the circuit operation are referred to references 5 and 6. Note that the simplest literals to form are the weak literals $d_t(X)$ followed in complexity by $U_t(X)$ then $D_t(X)$.

The other type of literal are the delta literals, X^I , which equal 1 only when $X = I$. Thus the strong delta literals are defined as:

*The margin by which zero must be exceeded will become clear when the circuits used to process the output of a weak literal circuit are presented.

** (cont'd.) literals are called literals by Allen.¹ The definitions given here are generalizations of this prior terminology.

$$X^I = 1 \text{ iff } X = I \quad (I = 0, 1, 2, 3) \quad (a)$$

$$X^I = 0 \text{ iff } X \neq I \quad (b) \quad (5)$$

For example, $X^2 = \langle 0, 0, 1, 0 \rangle$

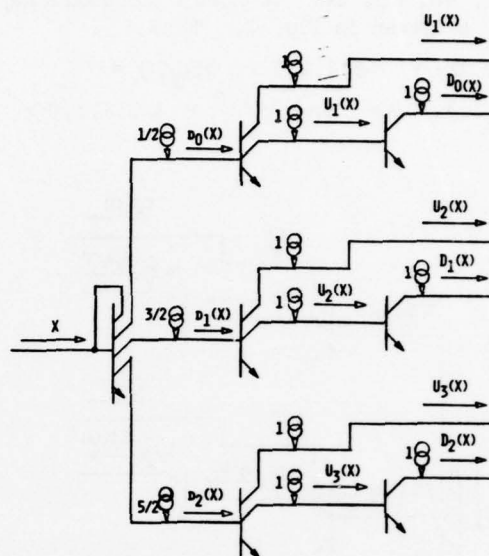


Figure 1

Circuit to Generate Threshold Literals

It is also possible to allow two superscripts which specify a range of values for which the value 1 is to be taken on. Thus:

$$X^{I,J} = 1 \text{ iff } I \leq X \leq J \quad (I, J = 0, 1, 2, 3) \quad (c)$$

$$X^{I,J} = 0 \text{ iff } I > X \text{ or } X > J \quad (d) \quad (5)$$

For example, $X^{1,2} = \langle 0, 1, 1, 0 \rangle$

The corresponding weak threshold literals are defined as:

$$x^I > 0 \text{ iff } x = I \quad (I = 0, 1, 2, 3) \quad (a)$$

$$x^I = 0 \text{ iff } x \neq I \quad (b) \quad (6)$$

$$x^{I,J} > 0 \text{ iff } I \leq x \leq J \quad (c)$$

$$x^{I,J} = 0 \text{ iff } I > x \text{ or } x > J \quad (d)$$

For example, $x^{1,2} = \langle 0, +, +, 0 \rangle$

The delta literals are derived from the threshold literals by use of the INHIBIT circuit to be described subsequently.

A straightforward, but important, generalization of the literals just defined permits them to take on values other than 1 when the appropriate conditions are satisfied. Thus we have:

$$\begin{aligned}
CU_t(X) &= C \text{ iff } X \geq t & (a) \\
CD_t(X) &= C \text{ iff } X \leq t & (b) \\
CX^I &= C \text{ iff } X = I & (c) \\
CX^{I,J} &= C \text{ iff } I \leq X \leq J & (d)
\end{aligned}
\tag{7}$$

with the conditions for 0 unchanged from equations 3b, 4b, and 5b. A circuit for producing CU_t and CD_t is given in Fig. 2. Thus,

$$\begin{aligned}
2U_2(X) &= \langle 0, 0, 2, 2 \rangle, \quad 3D_2(X) = \\
&\langle 3, 3, 3, 0 \rangle, \quad \text{and } 3X^{1,2} = \langle 0, 3, 3, 0 \rangle.
\end{aligned}$$

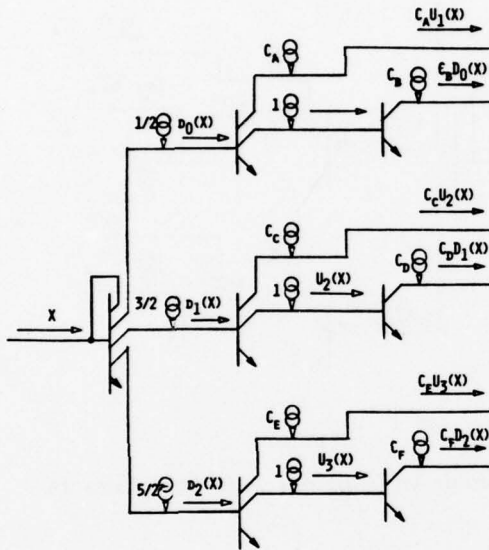


Figure 2

Circuit for Generalized Threshold Literals

COMPLEMENT

A function which occurs naturally in MVI^2L circuits is the complement in which the value of X is subtracted from a fixed bias, usually 3. A formal definition is:

$$\begin{aligned}
\bar{X}^B &= B \ominus X \triangleq B - X \text{ iff } B \geq X & (a) \\
\bar{X}^B &= B \ominus X \triangleq 0 \text{ iff } B \leq X & (b)
\end{aligned}
\tag{8}$$

If the B subscript is missing, it is taken to be 3: $\bar{X} = \bar{X}^3$. Thus if

$$\begin{aligned}
X &= \langle 0123 \rangle \\
\bar{X} &= \bar{X}^3 = \langle 3210 \rangle \\
\bar{X}^2 &= \langle 2100 \rangle \\
\bar{X}^1 &= \langle 1000 \rangle
\end{aligned}$$

A circuit to generate \bar{X}^B is shown in Fig. 3.

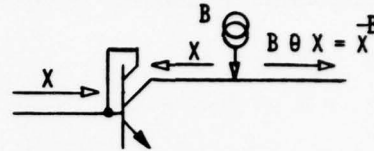


Figure 3. A Circuit to Generate \bar{X}^B

Table 1 illustrates these definitions of literals and complement. Inspection of this table verifies the validity of the interrelations among the various definitions presented in Table 2.

Table 1

Illustration of Literals and Complements

(a) Strong Literals

	X	0	1	2	3
\bar{X}		3	2	1	0
$X^{1,3} = D_0^0(X) = D_2(\bar{X}) = U_1(X)$		0	1	1	1
$X^0 = U_1^0(X) = U_3(\bar{X}) = D_0(X)$		1	0	0	0
$X^{2,3} = D_1^0(X) = D_1(\bar{X}) = U_2(X)$		0	0	1	1
$X^{0,1} = U_2^0(X) = U_2(\bar{X}) = D_1(X)$		1	1	0	0
$X^3 = D_2^0(X) = D_0(\bar{X}) = U_3(X)$		0	0	0	1
$X^{0,2} = U_3^0(X) = U_1(\bar{X}) = D_2(X)$		1	1	1	0
$D_0(X) = X^0$		1	0	0	0
X^1		0	1	0	0
X^2		0	0	1	0
$U_3(X) = X^3$		0	0	0	1
$X^{1,2}$		0	1	1	0

Table 1 (continued)

(b) Weak Literals

	X	0	1	2	3
\bar{X}		3	2	1	0
$d_2(\bar{X}) = u_1(X)$		0	+	+	+
$u_3(\bar{X}) = d_0(X)$		+	0	0	0
$d_1(\bar{X}) = u_2(X)$		0	0	+	+
$u_2(\bar{X}) = d_1(X)$		+	+	0	0
$d_0(\bar{X}) = u_3(X)$		0	0	0	+
$u_1(\bar{X}) = d_2(X)$		+	+	+	0

Note: + represents any base current large enough to saturate a transistor.

Table 2. Relations Among Literals

$$U_t(X) = D_{t-1}^\circ(X) = D_{3-t}(\bar{X})$$

$$D_t(X) = U_{t+1}^\circ(X) = U_{3-t}(\bar{X})$$

$$u_t(X) = d_{3-t}(\bar{X})$$

$$d_t(X) = u_{3-t}(\bar{X})$$

In these tables the delta literal notation is extended to functions so that if f represents an arbitrary function, then f° is equal to 1 only when f is equal to 0 and is 0 otherwise.

For example: if $X = \langle 0 \ 1 \ 2 \ 3 \rangle$

and $f(X) = \langle 3 \ 0 \ 1 \ 0 \rangle$

then $f^\circ(X) = \langle 0 \ 1 \ 0 \ 1 \rangle$

CONNECTIVES

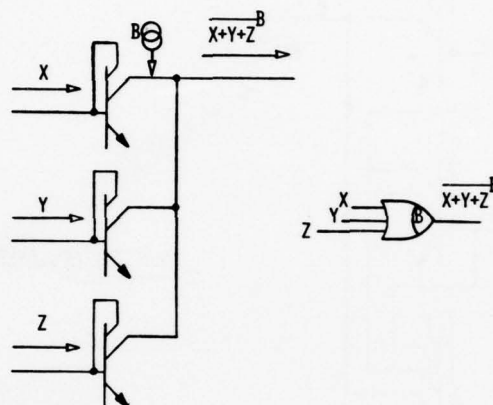
A simple means for combining two variables is to add the corresponding currents and then subtract the resulting sum current from a fixed bias current. A circuit which implements this operation is shown in Fig. 4. The resulting connective is called PLUS and is defined as follows:

PLUS Connective

$$\overline{X+Y+Z} \triangleq 3 \ominus (X+Y+Z) \quad (9a)$$

$$\overline{X+Y+Z}^B \triangleq B \ominus (X+Y+Z) \quad (9b)$$

Note: $X+Y+Z$ is the arithmetic sum of X and Y and Z ; the symbol \ominus is defined in equation (8).



(a) Circuit

(b) Logic symbol

Figure 4

Circuit for $\overline{X+Y+Z}^B$

Table 3 shows a table of values for $\overline{X+Y}$.

Table 3. Values of $\overline{X+Y}$

		X			
		0	1	2	3
Y	0	3	2	1	0
	1	2	1	0	0
	2	1	0	0	0
	3	0	0	0	0

$\overline{X+Y}$

By modifying slightly the circuit for PLUS it is possible to obtain a realization for the MAX connective. MAX selects the maximum value among the inputs and subtracts this value from a fixed bias. Formally:

MAX Connective

$$\overline{XvYvZ} \triangleq 3 \ominus \text{Maximum of } (X,Y,Z) \quad (10a)$$

$$\overline{XvYvZ}^B \triangleq B \ominus \text{Maximum of } (X,Y,Z) \quad (10b)$$

A circuit for \overline{MAX} is shown in Fig. 5 and a table of values for \overline{XvY} is given in Table 4.

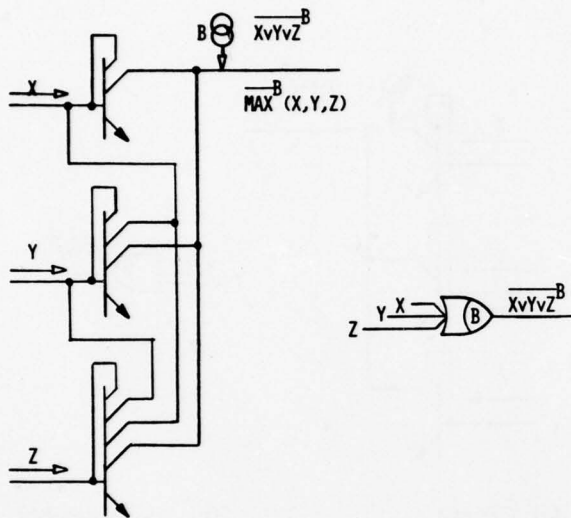


Figure 5
Circuit for \overline{XvYvZ}^B

Table 4. Values of \overline{XvY}

		X			
		0	1	2	3
Y	0	3	2	1	0
	1	2	2	1	0
	2	1	1	1	0
	3	0	0	0	0
		\overline{XvY}			

Situations can arise when it is desirable to calculate the maximum or minimum value of a number of variables. The maximum function is easily obtained by taking the complement of MAX. Thus

$$\text{Maximum of } (X,Y,Z) = \overline{\overline{MAX}(X,Y,Z)} \quad (11)$$

The minimum is derived by means of the following expression which can be verified by substituting all possible values (0,1,2,3) for the

variables. *

$$\text{Minimum of } (X,Y) = \overline{MAX}(\overline{X},\overline{Y}) \quad (12)$$

The third basic connective to be used is called INHIBIT. Whenever an inhibit input has a non-zero value, the output is forced to zero. A formal definition is:

INHIBIT

$$BX^{\circ}Y^{\circ}Z^{\circ} \triangleq B \text{ when } X = Y = Z = 0 \quad (13a)$$

$$BX^{\circ}Y^{\circ}Z^{\circ} \triangleq 0 \text{ when } X \text{ or } Y \text{ or } Z \neq 0 \quad (13b)$$

(B represents a fixed bias and if the inputs X, Y or Z are literals, either strong or weak literals can be used.) Figure 6 shows an INHIBIT circuit and Table 5 lists the conditions for a two variable INHIBIT operation.

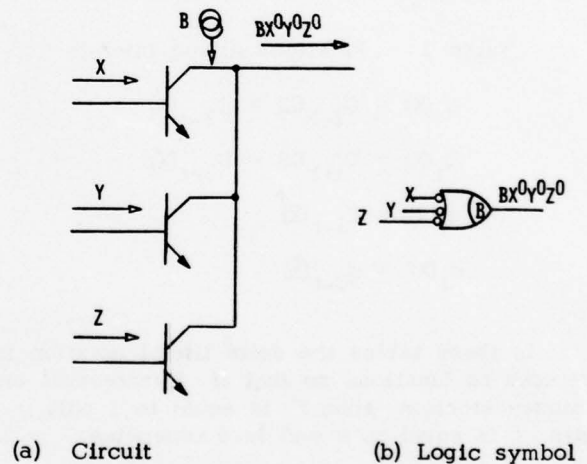


Figure 6
Circuit for $BX^{\circ}Y^{\circ}Z^{\circ}$

*Many algebraic systems for multi-valued logic use the minimum and maximum operators as basic connectives. If XvY is used to represent the maximum of X and Y, and $X\wedge Y$ is used to represent the minimum of X and Y, then equation 12 can be written as: $X\wedge Y = \overline{XvY}$. The similarity to De Morgan's Law for Boolean Algebras is interesting to note.

Table 5. Values of $BX^{\circ}Y^{\circ}$

		X			
		0	1	2	3
Y	0	B	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0

$BX^{\circ}Y^{\circ}$

One use of the INHIBIT circuit is in generating the delta literals (defined previously). Thus:

$$BX^1 = BD_0^{\circ}U_2^{\circ} \quad (14a)$$

$$BX^2 = BD_1^{\circ}U_3^{\circ} \quad (14b)$$

$$BX^{1,2} = BD_0^{\circ}U_3^{\circ} \quad (14c)$$

In forming the delta literals by means of the INHIBIT function, weak threshold literals can be used in place of strong threshold literals. Other possible valid expressions for the delta literals include

$$BX^1 = Bd_0^{\circ}U_2^{\circ} = BD_0^{\circ}u_2^{\circ}, \text{ etc. In general:}$$

$$X^{I,J} = d_{I-1}^{\circ}u_{J+1}^{\circ} = d_{I-1}^{\circ}d_J = u_{I+1}^{\circ}u_{J+1}^{\circ} \quad (15)$$

where it is understood that $X^{I,I} = X^I$. Particularly efficient realizations result from the expressions $X^2 = d_1^{\circ}\bar{X}$ and $X^1 = d_0^{\circ}\bar{X}^2$ which are discussed in the following section.

The three connectives (MAX, PLUS, and INHIBIT) just described can be combined in a single gate as shown in Fig. 7.* The subsequent discussions will deal only with use of the Literals and the Universal Gate; the connectives and complement function previously discussed can all be realized with the Universal Gate by omitting the input connections not required.

It has been proven⁴ that any multivalued function can be realized with the literals and the Maximum and Minimum connectives. Since those connectives can be realized with the Universal Gate, it follows that by using the Universal Gate circuit and the Literal circuits any multivalued function can be realized.

*This figure shows two inputs of each type. In general, up to three W_i inputs are allowed. There are no specific limitations on the numbers of X_i and Y_i inputs other than general circuit considerations of margins, capacitances, etc.

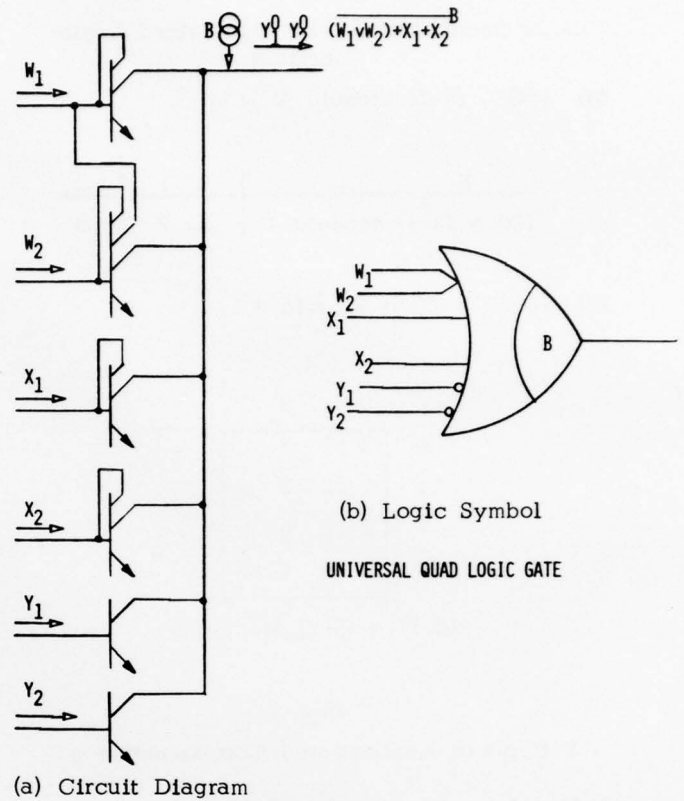


Figure 7

SINGLE-INPUT CIRCUITS

Techniques for designing multi-valued I^2L logic circuits will be presented by first considering the case of single-input circuits and then generalizing to more inputs. It may seem strange to consider single-input logic circuits, since they are trivial for the binary case; however, single-input multi-valued circuits illustrate many of the important issues of the general design problem.

Notation. Any multi-valued logic function can be specified by listing in a table the function's value for each combination of values of the input variables.** Thus Table 6a shows the specification for the function $f(X) = (X+1) \text{ Modulo } 4$, and Table 6b shows the specification for $f(X,Y) = (X+Y) \text{ Modulo } 4$. For single-variable functions the abbreviated notation introduced earlier will be used. In this notation the function is specified by listing its values between a pair of brackets: Thus $\langle 1230 \rangle$ is the specification for the function $f(X) = X+1 \text{ (Modulo } 4)$ of Table 6a.

**For binary functions such a table is called a Table of Combinations or Truth Table.

The first value represents $f(0)$, the second value $f(1)$, etc.

Table 6

Tabular Specifications for Four-Valued Logic Functions

(a) $f(X) = (X+1)$ Modulo 4

X	0	1	2	3
$f(X) = (X+1)$ Modulo 4	1	2	3	0

(b) $f(X,Y) = (X+Y)$ Modulo 4

		X			
		0	1	2	3
Y	0	0	1	2	3
	1	1	2	3	0
	2	2	3	0	1
	3	3	0	1	2

$f(X,Y) = (X+Y)$ Modulo 4

Table 7

Patterns of 0-entries and Corresponding p_i

Pattern**	p_i	Pattern**	p_i
<0--->	d_0	<-00->	$x^{1,2}$
<-0-->	x^1	<-0-0>	$x^1 u_3$
<--0->	x^2	<--00>	u_2
<---0>	u_3	<000->	d_2
<00-->	d_1	<00-0>	$d_1 u_3$
<0-0->	$d_0 x^2$	<0-00>	$d_0 u_2$
<0--0>	$d_0 u_3$	<-000>	u_1

**A - indicates a non-zero entry

Design Procedure. The design procedure realizes any single-variable function with one Universal Gate having appropriate literal inputs. The first step involves identifying the Inhibit inputs.

Step 1

- (a) Rewrite the Function $f(X)$ in the form $p_1^0 p_2^0 \dots p_n^0 f^*(X)$ where each of the p_i correspond to zeros of the function, and the p_i are weak (or strong) literals. Table 7 lists all 14 possible patterns of 0-entries and the corresponding choices for the p_i . Non-zero entries in

the literal term will produce zero entries in the f function.

- (b) Set $f^*(X)$ equal to $f(X)$ with each of the 0-entries replaced by \emptyset . The \emptyset -entries are "don't cares" which are allowed to take on any value. Combining f^* with the p_i guarantees that $f(x)$ will be zero for each of the \emptyset -entries. If there are no 0-entries in f , then $f = f^*$.

Examples

$$\begin{aligned} f_1(X) &= \langle 3120 \rangle = u_3^0 \langle 312\emptyset \rangle \\ \text{where } p_1 &= u_3 \text{ and } f_1^* = \langle 312\emptyset \rangle \\ f_2(X) &= \langle 0121 \rangle = d_0^0 \langle \emptyset 121 \rangle \\ f_3(X) &= \langle 2013 \rangle = (x^1)^0 \langle 2\emptyset 13 \rangle \\ f_4(X) &= \langle 3001 \rangle = (x^{1,2})^0 \langle 3\emptyset \emptyset 1 \rangle \\ f_5(X) &= \langle 3100 \rangle = u_2^0 \langle 31\emptyset \emptyset \rangle \\ f_6(X) &= \langle 1213 \rangle = \langle 1213 \rangle \\ f_7(X) &= \langle 1221 \rangle = \langle 1221 \rangle \\ f_8(X) &= \langle 0101 \rangle = d_0^0 (x^2)^0 \langle \emptyset 1\emptyset 1 \rangle \end{aligned}$$

The next step involves forming the complement of f^* since the Universal Gate has a complemented output.

Step 2. Rewrite f^* as $\overline{f^{**B}}$ where B is the largest entry of f^* .

Examples

$$\begin{aligned} f_1^* &= \langle 312\emptyset \rangle = \langle \overline{021\emptyset} \rangle; f_1^{**} = \langle 021\emptyset \rangle \\ f_2^* &= \langle \emptyset 121 \rangle = \langle \overline{\emptyset 101} \rangle^2; f_2^{**} = \langle \emptyset 101 \rangle \\ f_3^* &= \langle 2\emptyset 13 \rangle = \langle \overline{1\emptyset 20} \rangle; f_3^{**} = \langle 1\emptyset 20 \rangle \\ f_4^* &= \langle 3\emptyset \emptyset 1 \rangle = \langle \overline{0\emptyset \emptyset 2} \rangle; f_4^{**} = \langle 0\emptyset \emptyset 2 \rangle \\ f_5^* &= \langle 31\emptyset \emptyset \rangle = \langle \overline{02\emptyset \emptyset} \rangle; f_5^{**} = \langle 02\emptyset \emptyset \rangle \\ f_6^* &= \langle 1213 \rangle = \langle \overline{2120} \rangle; f_6^{**} = \langle 2120 \rangle \\ f_7^* &= \langle 1221 \rangle = \langle \overline{1001} \rangle^2; f_7^{**} = \langle 1001 \rangle \\ f_8^* &= \langle \emptyset 1\emptyset 1 \rangle = \langle \overline{\emptyset 0\emptyset 0} \rangle^1; f_8^{**} = \langle \emptyset 0\emptyset 0 \rangle \end{aligned}$$

The design is completed by expressing f^{**} as a sum of strong literals.

Step 3. The procedure for decomposing f^{**} is explained most easily by considering the patterns of non-zero entries in f^{**} . First it should be noted that there will always be at least one zero entry because f^{**} is formed by taking the complement of f^* with respect to its largest entry.

- (a) If there is exactly one non-zero entry in f^{**} ($\langle -000 \rangle$, $\langle 0-00 \rangle$, $\langle 00-0 \rangle$ or $\langle 000- \rangle$), then

f^{**} is equal to the corresponding strong literal. For example, $f^{**} = \langle 0200 \rangle = 2X^1$, $f^{**} = \langle 0003 \rangle = 3U_3$.

- (b) If there are two non-adjacent, non-zero entries ($\langle -0-0 \rangle$, $\langle -00- \rangle$ or $\langle 0-0- \rangle$), then f^{**} is the PLUS of the corresponding strong literals. For example,

$$f^{**} = \langle 1020 \rangle = D_0 + 2X^2,$$

$$f^{**} = \langle 3002 \rangle = 3D_0 + 2U_3$$

- (c) Suppose that f^{**} is made up of a string of adjacent non-zero entries such as $\langle XX00 \rangle$, $\langle XXX0 \rangle$, $\langle 0XX0 \rangle$, $\langle 0XXX \rangle$, $\langle 00XX \rangle$. If all entries of the string have the same value, they are realized with one strong literal: $f^{**} = \langle 2200 \rangle = 2D_1$, $f^{**} = \langle 0110 \rangle = X^1, 2$. If not, f^{**} is decomposed into the PLUS of two functions, $f^{**} = g + h$, where g has the same zero's as f^{**} and has the minimum value of the string of f^{**} for all of its non-zero entries: $f^{**} = \langle 3200 \rangle = \langle 2200 \rangle + \langle 1000 \rangle = g + h$. The other function h is equal to $f^{**} - g$. The function g is realized with a single literal as described above. The other function, h , is realized as if it were an f^{**} function (using the rules given here) and then added to g . For example, suppose that $f^{**} = \langle 2310 \rangle$. Then $g = \langle 1110 \rangle = D_2$ and $h = \langle 1200 \rangle = \langle 1100 \rangle + \langle 0100 \rangle = D_1 + X^1$. Finally, $f^{**} = D_2 + D_1 + X^1$.

- (d) The final possibility is that f^{**} is composed of two strings of non-zero entries: either $\langle XX0X \rangle$ or $\langle X0XX \rangle$. Then the single non-zero entry is realized with a single literal and the string of two non-zero entries are realized as in (c) above. For example, if $f^{**} = 2013$, then $f^{**} = \langle 2000 \rangle + \langle 0011 \rangle + \langle 0002 \rangle = 2D_0 + U_2 + 2U_3$.

Because of the way in which it is derived, the f^{**} function often contains don't-care (\emptyset) entries. These should be specified so as to result in the fewest terms in the final f^{**} expression. If a choice is possible between a threshold or a delta literal, the threshold literal should usually be chosen.

[†] It should be pointed out that another valid expression for f^{**} would be $f^{**} = \text{MAX} \{ \langle 1110 \rangle, \langle 2200 \rangle, \langle 0300 \rangle \} = \text{MAX} \{ D_2, 2D_1, 3X^1 \}$. The realization with PLUS is preferred, since it uses fewer collectors. In fact single-variable functions can always be realized without using the MAX connective.

Table 8 shows the final expressions obtained by using the procedure just described on the Example functions. The last function, f_8 , is somewhat special in that $f_8^*(X) = \langle \emptyset 1 \emptyset 1 \rangle$ which can be made equal to the constant 1 function by setting both \emptyset 's equal to 1.

Table 8

Design Expressions for Single Variable Example Functions

$$\begin{aligned} f_1(X) &= \langle 3120 \rangle = u_3^0 \langle 3120 \rangle = u_3^0 \langle \overline{0210} \rangle = u_3^0 [\overline{U_1 + X^1}] \\ f_2(X) &= \langle 0121 \rangle = d_0^0 \langle \emptyset 121 \rangle = d_0^0 \langle \emptyset 101 \rangle^2 = d_0^0 [\overline{D_1 + U_3}]^2 \\ f_3(X) &= \langle 2013 \rangle = (X^1)^0 \langle 2013 \rangle = (X^1)^0 \langle \overline{1020} \rangle = (X^1)^0 [\overline{D_2 + X^2}] \\ f_4(X) &= \langle 3001 \rangle = (X^1, 2)^0 \langle 3001 \rangle = (X^1, 2)^0 \langle \overline{0002} \rangle = (X^1, 2)^0 [\overline{2U_3}] \\ f_5(X) &= \langle 3100 \rangle = u_2^0 \langle 3100 \rangle = u_2^0 \langle \overline{0200} \rangle = u_2^0 [\overline{2U_1}] \\ f_6(X) &= \langle 1213 \rangle = \langle 1213 \rangle = \langle \overline{2120} \rangle = [\overline{D_2 + D_0 + X^2}] \\ f_7(X) &= \langle 1221 \rangle = \langle 1221 \rangle = \langle \overline{1001} \rangle^2 = [\overline{D_0 + U_3}]^2 \\ f_8(X) &= \langle 0101 \rangle = d_0^0 (X^2)^0 \langle \emptyset 1 \emptyset 1 \rangle = d_0^0 (X^2)^0 \end{aligned}$$

SUMMARY AND CONCLUSIONS

The logic design of multi-valued I²L circuits has been discussed. A formalism for design was introduced and illustrated with a method for designing single-input circuits. Because of space limitations, the extension to more general circuits has not been included.

ACKNOWLEDGEMENTS

The original circuit designs were developed by T. Dao and L. K. Russell of Signetics. They and I jointly invented the circuit family described here. Thanks are due to L. K. Russell for his helpful comments on this manuscript.

REFERENCES

- Allen, C.M. and Givone, D.D., "A Minimization Technique for Multiple-Valued Logic Systems," *IEEE Trans. Computers*, pp. 182-184, Feb. 1968.
- Nagle, Jr., H.T., Carroll, B.D., and Irwin, J.D., "An Introduction to Computer Logic," Prentice-Hall, Inc. 1975.
- Vranesic, Z.G., Lee, E.S., and Smith, K.C., "A Many-Valued Algebra for Switching Systems," *IEEE Trans. Computers*, pp. 964-971, Oct. 1970.

- 4 Ying, C. and Susskind, A.K., "Building Blocks and Synthesis Techniques for the Realization of M-ary-Combinational Switching Functions," Conference Record 1971 Symposium on Theory and Applications of Multiple-Valued Logic Design, pp. 183-205.
- 5 Dao, T.T., Russell, L.K., Preedy, D.R., and McCluskey, E.J., "Multilevel I²L with threshold gates," in ISSCC Dig. Tech. Papers, Feb. 1977, p. 110.
- 6 Dao, T.T., McCluskey, E.J., and Russell, L.K., "Multivalued Integrated Injection Logic," IEEE Trans. Computers, pp. 1233-1241, Dec. 1977, Vol. C-26, No. 12.

SOME I^2L CIRCUITS FOR MULTIPLE-VALUED LOGIC

James H. Pugsley and Charles B. Silio, Jr.
Electrical Engineering Department
University of Maryland
College Park, Maryland 20742

Abstract

While various integrated circuit technologies have been proposed for multiple-valued logic, it appears that for the near future current-mode integrated injection logic (I^2L) will predominate. The present paper suggests I^2L circuits for the various gate types and design approaches that have been proposed, including some memory elements, and discusses some of the problems of using I^2L multiple-valued logic chips in practical designs.

I. Introduction

For the past several years there has been increasing interest in multiple-valued logic design. With the announcement [1] of commercial implementation we are entering the second stage in the cycle, where considerations of practical implementation must be dealt with. While various integrated circuit technologies have been proposed for implementation [2, 3, 4, 5] of multiple valued logic, it appears that for the near future I^2L will predominate.

A number of operator sets or gate types have been proposed for logic design [6, 7, 8], all of which have at least reasonable implementations in I^2L . The final choice of gate types will thus be strongly dependent on the development of practical design algorithms and packaging considerations, as well as on the basic gate implementations themselves. I^2L packaging and layout problems are somewhat different for large-scale integration (LSI) than for small- and medium-scale integration (SSI and MSI) due to the current-mode nature of multiple-valued I^2L . Because logic signals are represented by currents rather than by voltage levels, each output signal pin (or connection) has a fan-out of one! In the design of an LSI chip this limitation does not present serious problems, because the necessary replication of signals is easily accomplished at the input stage of each gate structure. Even with the assumption that most multiple-valued designs will use LSI chips, however, there will still be the need to provide interface logic to adapt the LSI chips to the intended application. Such interfacing will make use of SSI and MSI multiple-valued logic circuits, and it

is here that the fan-out limitation raises problems of what to package in a chip and how to design with such chips.

Section II discusses some considerations for I^2L circuits using current-mode multiple-valued logic. Section III presents suggested circuits for most of the multiple-valued logic operations that have been proposed. A brief discussion of multiple-valued memory elements in presented is Section IV.

II. I^2L Circuit Considerations

The three fundamental circuit operations in current-mode multiple-valued I^2L are the replication of signals using current mirrors, linear summation, and the detection of threshold values [9]. I^2L is a bipolar technology that can be fabricated using existing T^2L production equipment [9]. The transistors are operated in an inverted mode, so that multiple-collector transistors are the most common circuit element. No resistors are required, fixed-value current sources can easily be included in the structures with the consumption of little additional chip area, and very good speed-power products have been reported [10].

Since using currents to represent logical values results in a fan-out of one, replication of current signals is a common on-chip requirement. This is accomplished by means of a current-mirror, as shown in Figure 1. If the current gains for the individual collectors are given by $\beta_1, \beta_2,$

etc., then $i_{c_1} = \frac{\beta_1}{(\beta_1+1)} i$, $i_{c_2} = \frac{\beta_2}{(\beta_1+1)} i$, $i_{c_3} = \frac{\beta_3}{(\beta_1+1)} i$,

etc. In practice it is possible to make the area of collector c_1 just slightly smaller than that of the others, so that $i_{c_2} = i_{c_3} = \dots = i$. The term

"current-mirror" results from the fact that the polarity of the current i is reversed in the replicated copies. For the remainder of this paper it will be assumed that the magnitude of current in each "free" collector of a current mirror is equal to the input current.

Linear summation is provided simply by connecting together leads carrying current-mode signals. For example, to obtain a current (sink) signal with a value of $2x_1 + x_2$, the circuit of

Figure 2 will suffice.

The third major circuit operation is that of thresholding, which arises in the implementation of most of the unary operations proposed for multiple-valued logic. For setting thresholds the attractiveness of I^2L lies in the ease with which constant current sources can be integrated into the gate structure. A typical circuit configuration is shown in Figure 3, where the function implemented is given in the figure, and it is assumed that all input and output current signals take one of the discrete values $0, 1, 2, \dots, m-1$. Note that in this circuit one of the transistors is used as a simple (current-controlled) switch rather than as a current-mirror. The current source with value $3/2$ sets the threshold for the transistor switch, and the current source with value one provides the output current, which is shorted to ground by the transistor switch when the input current x is less than or equal to one.

In designing I^2L circuits to implement multiple-valued switching systems it is frequently necessary to convert sinks to current sources with the same value, and vice versa. Clearly, the simple current-mirror will perform a source to sink conversion. A sink to source conversion circuit is shown in Figure 4, where, as usual, logic signals take one of the values $0, 1, 2, \dots, m-1$. In this circuit the actual values of the current sources are not important provided they are both equal to each other and greater than or equal to the value of the highest possible signal current.

Solutions to the problem of unity fan-out are somewhat dependent on the choice of gate types (operators) implemented. This problem will be discussed below following the consideration of gates and their packaging.

III. Suggested I^2L Circuits

The logic design of multiple-valued switching systems presupposes that some decisions have been made as to the basic gates or operators that will be used. Several different choices have been proposed in the literature, and most of these will be discussed below. Unfortunately, different choices of basic gate types lead to different design procedures; there is no uniform design technique that will show which set of gates is best for a given system. Most of the gate types that have been proposed have reasonable implementations in current-mode I^2L . As a result, the choice of which gates (or which algebraic system) to use will usually be based on the ease of using the associated logic design tools.

Post Monotone Unary Operators

The Post algebra monotone unary operators $D_i(x)$ ($i = 1, 2, \dots, m-1$) and their complements can be implemented by simple threshold networks of the type shown in Figure 3. The monotone opera-

tors are defined by

$$D_i(x) = \begin{cases} m-1 & \text{if } x \geq i \\ 0 & \text{if } x < i \end{cases}, \text{ and}$$

$$\overline{D}_i(x) = \begin{cases} 0 & \text{if } x \geq i \\ m-1 & \text{if } x < i \end{cases}.$$

Circuits for these operators are shown in Figure 5.

Post Disjoint Unary Operators

The Post algebra disjoint operators $C_i(x)$ ($i = 0, 1, 2, \dots, m-1$) each requires two thresholds for its implementation. These operators are defined by

$$C_i(x) = \begin{cases} m-1 & \text{if } x = i \\ 0 & \text{if } x \neq i \end{cases},$$

and can be implemented by the circuit of Figure 6. This circuit is, in effect, a combination of a circuit for $D_i(x)$ and one for $\overline{D}_{i+1}(x)$, reflecting the Post algebra identity $C_i(x) = D_i(x) \wedge \overline{D}_{i+1}(x)$. (In the Post algebra with elements $\{0, 1, \dots, m-1\}$ the " \wedge " operation is just "minimum".) Note that a "wired-min" operation is achieved in Figure 6 by connecting the collectors of the two transistor switches.

Literals

Another set of unary operators based on Post algebra is the "literals" defined in [6]. The usual notation for a literal in the variable x is a_x^b , defined by:

$$a_x^b = \begin{cases} m-1 & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}.$$

Inspection of the definitions will readily show that $a_x^b = D_a(x) \wedge \overline{D}_{b+1}(x)$, and will lead to the circuit of Figure 7 with its obvious similarity to the $C_i(x)$ implementation that was shown in Figure 6. Note also that $i_x^i = C_i(x)$, $i_x^{m-1} = D_i(x)$ and $0_x^{i-1} = \overline{D}_i(x)$.

Multiplexers

The basic circuit structures of current-mirrors threshold detectors, and shorting switches can be combined to implement an m -line to one-line multiplexer for m -valued signals. Such a multiplexer can be used as a single-variable universal logic module (ULM), and several of them can be cascaded to implement any given function of any number of variables. Such a tree-type implementation may lead to excessive pin counts for functions of many variables, but it is often quite useful for reasonably small numbers of variables. A circuit to implement a multiplexer for $m = 3$ (3-valued logic) is shown in Figure 8. As usual, all of the variables in this circuit are represented by 3-valued

currents. The two sources and the current-mirror at the output perform a sink-to-source conversion. The three current-mirrors driving this converter have their outputs tied together since at any given time only one of the three collectors can have a non-zero current.

Max-gates

All of the proposed operator sets that are based on Post algebra require the use of max and min operations. A two-input max-gate is shown in Figure 9, where once again the output section of the circuit performs a sink-to-source conversion. Thus a max-gate for m -valued logic will serve as a max-gate for k -valued logic for any $k \leq m$.

The basic structure of the circuit in Figure 9 can be extended to implement max-gates with more than two inputs. A three-input max-gate is shown in Figure 10.

Pseudo-complement

Before discussing circuits to implement the min operation, it is necessary to consider the unary operation defined by $x' = (m-1) - x$, which is a generalization of the complement operation in 2-valued logic. We will refer to this operation as "pseudo-complement" because only the elements 0 and $m-1$ possess true (lattice) complements. The pseudo-complement operation is closely related to sink-to-source conversion, as can be seen from the circuit implementation shown in Figure 11.

The set of operations {max, min, pseudo-complement} is not sufficient to implement every possible function, except for when $m=2$. (For $m=2$, the resulting Post algebra reduces to the familiar 2-valued switching algebra, with max = OR, min = AND, and ' being the usual complement.) Hence, for $m > 2$ some other unary operators, such as literals, $C_i(x)$'s, $D_i(x)$'s, etc., are required for functional completeness. However, the pseudo-complement operation is sufficiently useful that it is well worth including it in the set of available operators.

Before leaving the pseudo-complement operation, it should be noted that there are frequent occasions on which the logic of a design calls for a max-gate driving a pseudo-complement. From the circuits in Figures 9 and 11 it should be apparent that the pseudo-complement can be implemented by simply removing the final current source and the current-mirror from the max-gate circuit, resulting in the generalized "max'-gate" similar to that in [1]. A two input max'-gate with a fan-out of two is shown in Figure 12.

Min-gates

The circuit for a min-gate is derived from the algebraic identity $x \wedge y = (x' \vee y')$, and is a combin-

ation of two pseudo-complement circuits and a max'-gate, as shown in Figure 13.

Modular Algebra Operations

The only algebraic systems other than Post algebras to receive significant consideration have been modular algebras [11], in which the available operations are addition modulo- m and multiplication modulo- m on the set of elements $\{0, 1, 2, \dots, m-1\}$. These operators can be used to implement any given function if and only if the integer m is prime. While the resulting gate networks are, in general, much more complex than the corresponding ones using Post algebra operations, there are some classes of functions (notably those that arise in arithmetic operations) for which modular addition and modular multiplication seem well suited.

The circuit for a modulo- m adder is shown in Figure 14. While in concept this circuit is straightforward, it requires large-value $2(m-1)$ current sources which may cause fabrication difficulties.

The need for large-value current sources can be avoided by using multiplexers to implement the addition operation. Figure 15 shows a modulo-3 adder using the multiplexer (mux) modules of Figure 8. This approach can be used for any value of m and requires a current-mirror and $m-1$ sink-to-source conversion networks for fan-out expansion, and m m -valued multiplexer modules.

For modulo- m multiplication one possible solution is to use multiplexers as shown in block diagram form in Figure 16 for $m=3$. The box marked "fan-out" is identical to the input portion of Figure 15. Again, this network can be easily generalized to any value of m .

Two special cases of modular operations are of considerable importance. These are incrementation by one (mod- m) and multiplication by a constant (mod- m). The incrementation operation on x is sometimes denoted by x^* , and a circuit for performing the operation is shown in Figure 17. In this circuit the upper branch (after the current-mirror) is a sink-to-source converter for the signal $x+1$, and the lower branch threshold detects and forces the output current to zero if $x=m-1$.

This incrementation operation is sometimes called the "cycle" operation and was introduced originally by Post [12]. Its chief importance lies in the fact that the set of operators {max, min, cycle} is functionally complete for any value of m , and any function can be implemented using only these operations. (In fact, the sets {max, cycle} and {min, cycle} are each functionally complete.) The possibility of using only one type of unary operator is attractive, but, unfortunately, the resulting networks are in general so complex as to more than offset the advantages when compared to networks using C-operators, D-operators, or multiplexers.

The other special case, multiplication (modulo- m) by a constant, is of more practical importance

because such elements are one of the building blocks (along with modulo-m adders and storage elements) of m-valued linear sequential systems [13, 14]. Multiplication by a constant is easily achieved with a single multiplexer module, as shown in Figure 18.

IV. Multiple-valued Memory Elements

While it appears that the fabrication of high-density random-access memory (RAM) chips still presents a problem, memory elements for register and control functions are fairly easily obtainable. The basic multiple-valued memory element [15] in I^2L technology is a generalization of the binary NOR-gate latch and has the block diagram shown in Figure 19(a). The max'-gates used are two-output gates, as in the circuit in Figure 12. This memory circuit can be called an RS m-valued latch by analogy to the corresponding binary circuit. The chief difference from the binary case is that a quantizer network [16] to restore signal levels must be placed in each feedback loop. With such a quantizer the circuit will function properly so long as the loop gain L satisfies the restriction

$$\frac{m-3/2}{m-1} < L < \frac{m-3/2}{m-2}.$$

Since the loop gain is the product of the gain through one or more current mirrors (nominally equal to one) this constraint should be easily met for practical values of m. The logical diagram of a gated RS m-valued latch is shown in Figure 19(b). In an actual circuit it is impractical to use current-mode signals as clocks, due to fan-out difficulties, and so the gated latch circuit of Figure 19(c) assumes a voltage-level (negative) clock pulse, which allows implementing min-gates of Figure 19(b) with simple shorting switches as shown. As pointed out by Irving and Nagel [15], such a circuit has m stable states and can be made to undergo any desired state transition by application of the proper input signals. The Q' output is the pseudo-complement of the Q output. Here, as in the binary case, some input values produce critical races which should be avoided. For the m-valued RS latch the restrictions on the inputs to avoid indeterminate behavior are that $R+S < m$. A master-slave RS m-flop can be constructed as indicated in Figure 20. In this case the current source I must be large enough to saturate both of the transistors it drives. A resistor could be used in place of the current source, but in I^2L technology the current source is likely to be easier to fabricate.

As an example of the use of some of the circuits presented Figure 21 shows a 4,1 quaternary-coded decimal counter. Using the map synthesis technique for Post D-operators suggested by Ying and Susskind [17] gives the input equations

$$R_1 = D_2(Q_1) \wedge D_1(Q_2)$$

$$S_1 = 2 \wedge D_1(Q_1) \wedge D_3(Q_2) \vee 1 \wedge D_3(Q_2)$$

$$R_2 = D_2(Q_1) \wedge D_1(Q_2) \vee D_3(Q_2)$$

$$S_2 = D_2(Q_2) \wedge \bar{D}_3(Q_2) \vee 2 \wedge \bar{D}_2(Q_1) \wedge D_1(Q_2) \wedge \bar{D}_3(Q_2) \vee 1 \wedge \bar{D}_1(Q_2)$$

which are implemented in a manner closely related to the single D-operator circuits of Figure 5. The circuit shown requires significantly fewer transistors than would implementation of these same 4-flop input equations using multiplexer modules, in part due to the use of "wired-min" operations.

V. Conclusion

Application of the circuits suggested above awaits their fabrication and commercial availability as MSI and SSI chip sets. Implementation of I^2L circuits for values of $m > 4$ appears to depend primarily on the accuracy of the lithographic process used in their fabrication and on the ratio of the largest-to-smallest current source values used in the threshold detectors and other circuit components. Assuming these fabrication difficulties can be overcome successfully, an inherent shortcoming in the use of current-mode logic remains, namely, the unity fan-out capability of each signal lead. A major advantage of multiple-valued circuits is their potential for reducing the chip area consumed by interconnecting signal wires on-board LSI chips and for reducing the pin-count per package for the (more complex) logic signals entering and exiting the LSI chip. The set of MSI and SSI multiple-valued logic chips used to provide functional interfacing between both LSI component chips and the external environment must include modules for sink-to-source conversion, fan-out expansion, and (possibly programmable) constant current sources, if multiple-valued current-mode signals are to be used. As seen in Figures 15 and 16, this is the case even if multiplexers are used as tree-type universal logic modules. Because the complexity of the interfacing circuitry is dependent on both the particular logic design technique used and on the particular function being implemented, all of the circuits suggested above are considered appropriate for implementation in a family of MSI and SSI modules for each particular value of m.

References

1. T. T. Dao, L. K. Russell, D. R. Preedy and E. J. McCluskey, "Multilevel I^2L with Threshold Gates", Proc. 1977 IEEE Intl. Solid-State Circuits Conf., pp. 110-111 & 243.
2. K. C. Smith, "Circuits for Multiple Valued Logic--A Tutorial and Appreciation", Proc. Sixth Intl. Symp. on Multiple-Valued Logic, May 1976, pp. 30-43.
3. Z. G. Vranesic, K. C. Smith and A. Druzeta, "Electronic Implementation of Multi-Valued Logic Networks", Proc. 1974 Intl. Symp. on

Multiple-Valued Logic, pp. 59-78.

4. H. T. Mouftah and J. B. Jordan, "Integrated Circuits for Ternary Logic", Proc. 1974 Intl. Symp. on Multiple-Valued Logic, pp. 285-302.
5. A. Druzeta, A. S. Sedra and Z. G. Vranesic, "Multi-Valued Logic Circuits", Proc. 12th Annual Allerton Conf. on Circuit and System Theory, Oct. 1974, pp. 547-556.
6. C. M. Allen and D. D. Givone, "A Minimization Technique for Multiple-Valued Logic Systems", IEEE Trans. Computers, Vol. C-17, Feb. 1968, pp. 182-184.
7. Z. G. Vranesic, E. S. Lee and K. C. Smith, "A Many-Valued Algebra for Switching Systems", IEEE Trans. Computers, Vol. C-19, Oct. 1970, pp. 964-971.
8. O. Ishizuka, "Multi-Valued Multi-Threshold Networks", Proc. 1976 Intl. Symp. on Multiple-Valued Logic, pp. 44-47.
9. T. A. Longo, "Advances in Bipolar Logic for Computer Technology", Digest of Papers COMPCON Fall 77, Sept. 1977, pp. 245-246.
10. J. L. Stone, "I²L: A Comprehensive Review of Techniques and Technology", Solid State Technology, June 1977, pp. 42-48.
11. P. E. Wood, Jr., Switching Theory, McGraw-Hill, New York, 1968.
12. E. L. Post, "Introduction to a General Theory of Elementary Propositions", Am. Journ. Math., Vol. 43, 1921, pp. 163-185.
13. A. Gill, "Linear Modular Systems", Chapt. 5 in L. A. Zadeh and E. Polak, System Theory, McGraw-Hill, New York, 1969, pp. 179-231.
14. M. E. Conner and C. B. Silio, Jr., "Lattice Properties for Linear State Assignment over Residue Class Rings", Proc. Seventh Intl. Symp. on Multiple-Valued Logic, May 1977, pp. 64-69.
15. T. A. Irving and H. T. Nagle, "An Approach to Multi-Valued Sequential Logic", Conf. Rec. 1973 Symp. on Multiple-Valued Logic, pp. 89-105.
16. T. T. Dao, E. J. McCluskey and L. K. Russell, "Multivalued Integrated Injection Logic", IEEE Trans. Computers, Vol. C-26, Dec. 1977, pp. 1233-1241.
17. C. Ying and A. K. Susskind, "Building Blocks and Synthesis Techniques for the Realization of m-ary Combinational Switching Functions", Conf. Rec. 1971 Symp. on the Theory and Applications of Multiple-Valued Logic Design, pp. 183-205.

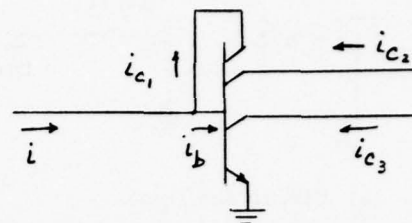


Figure 1. I²L Current mirror.

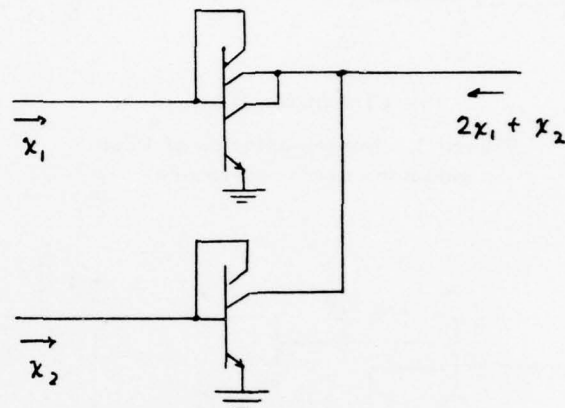


Figure 2. Example of linear summation.

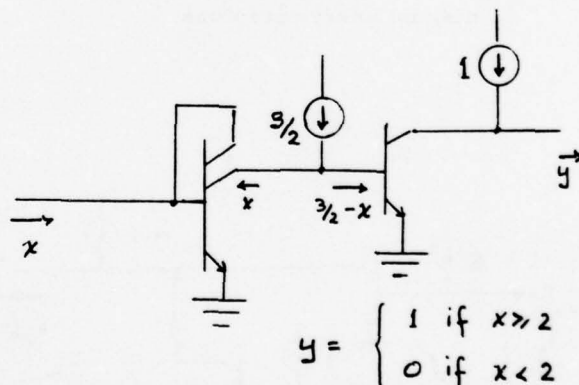


Figure 3. Thresholding example.

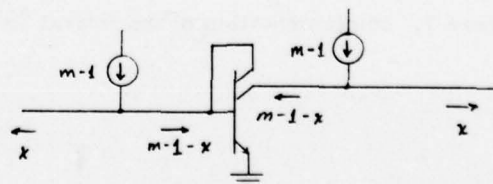
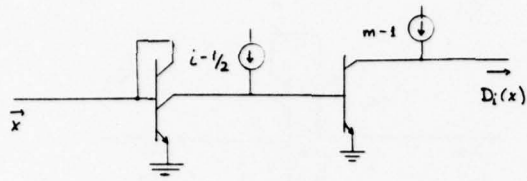
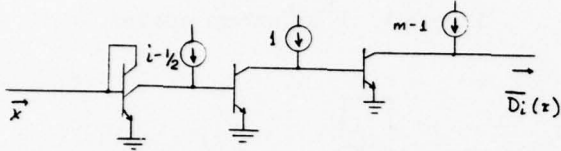


Figure 4. Sink-to-source conversion.



(a) Circuit for $D_i(x)$.



(b) Circuit for $\bar{D}_i(x)$.

Figure 5. Implementation of Post monotone unary operators.

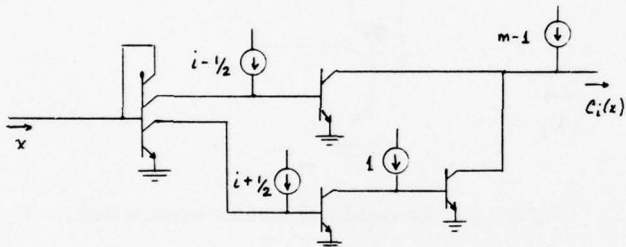


Figure 6. Implementation of Post disjoint unary operators.

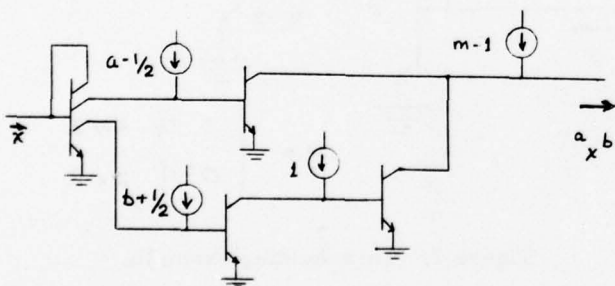
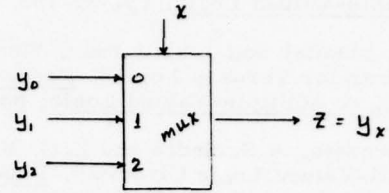
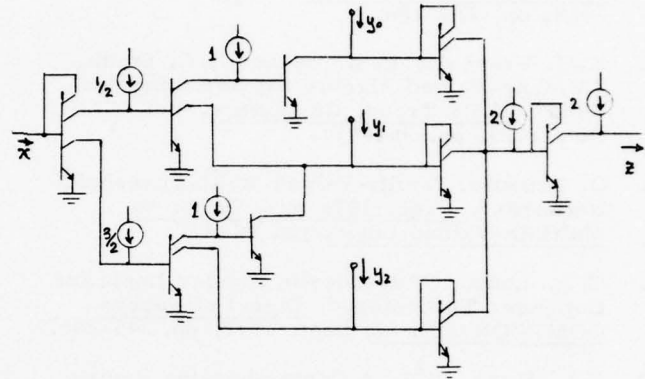


Figure 7. Implementation of the literal a_x^b .



(a) Block diagram.



(b) Circuit.

Figure 8. Three-to-one three-valued multiplexer.

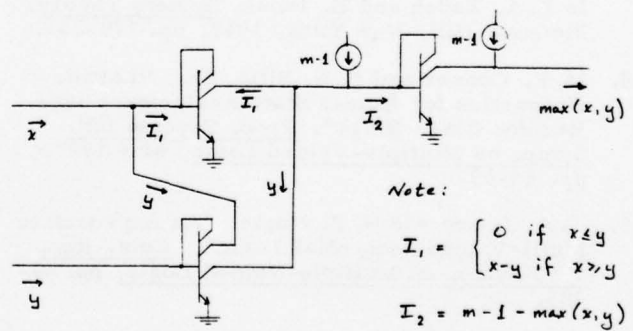


Figure 9. Two-input max-gate.

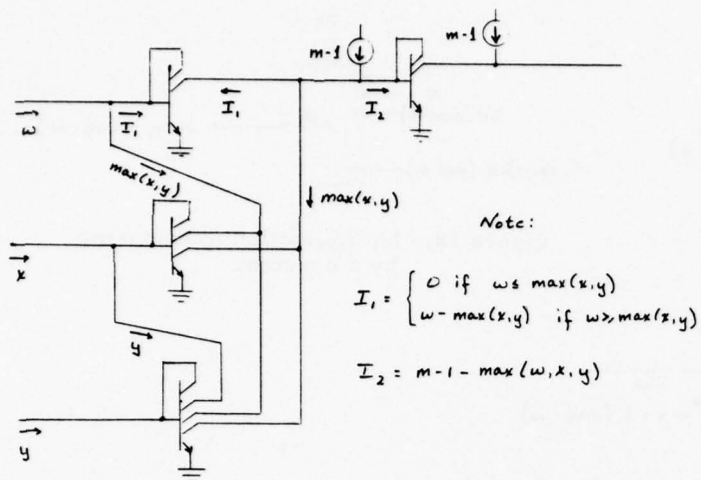


Figure 10. Three-input max-gate.

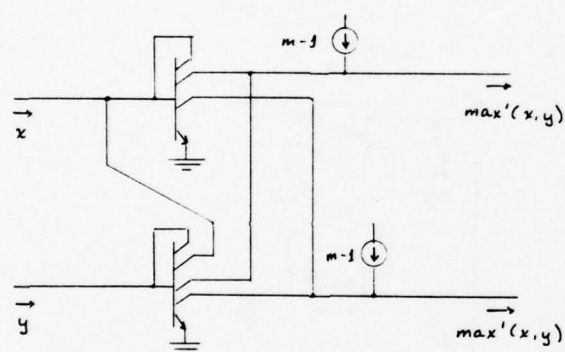


Figure 12. Two-input two-output max'-gate.

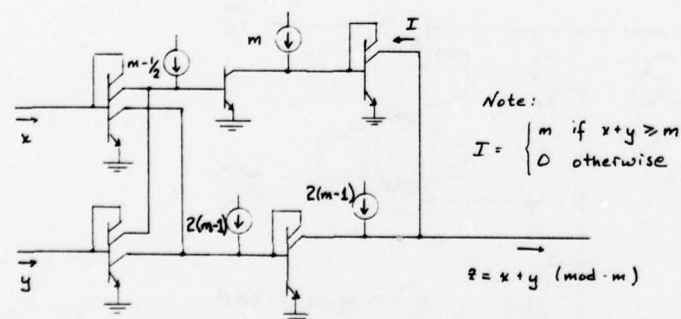


Figure 14. Modulo-m adder.

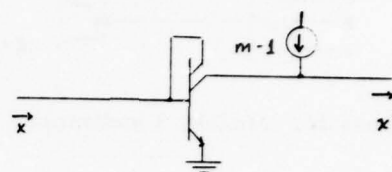


Figure 11. Pseudo-complement implementation.

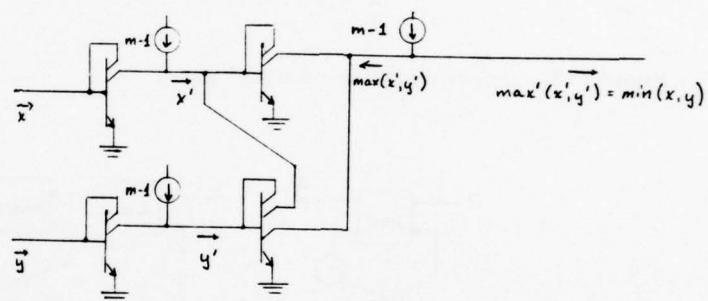


Figure 13. Two-input min-gate.

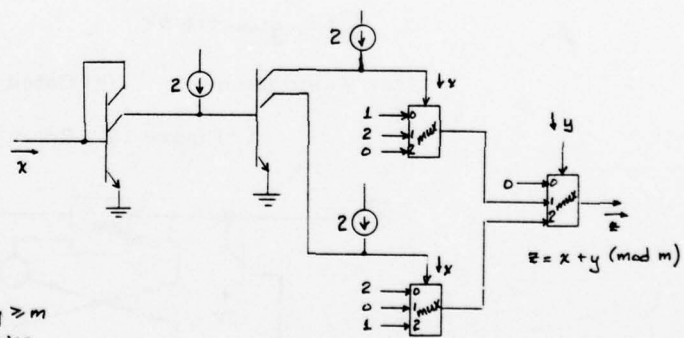


Figure 15. Modulo-3 adder using multiplexers.

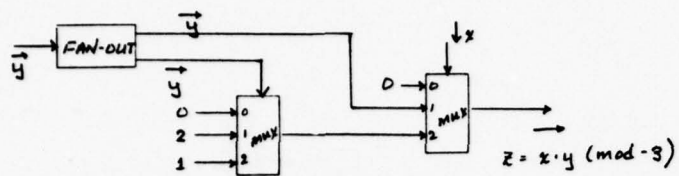


Figure 16. Modulo-3 multiplier.

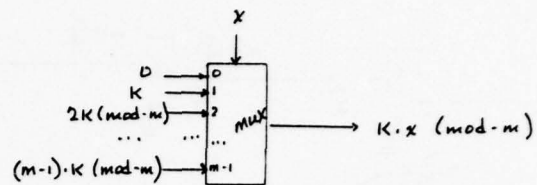


Figure 18. Multiplication (modulo-m) by a constant.

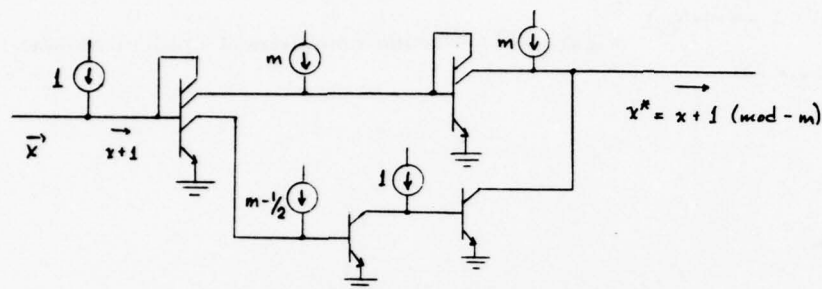


Figure 17. Incrementation or cycle gate.

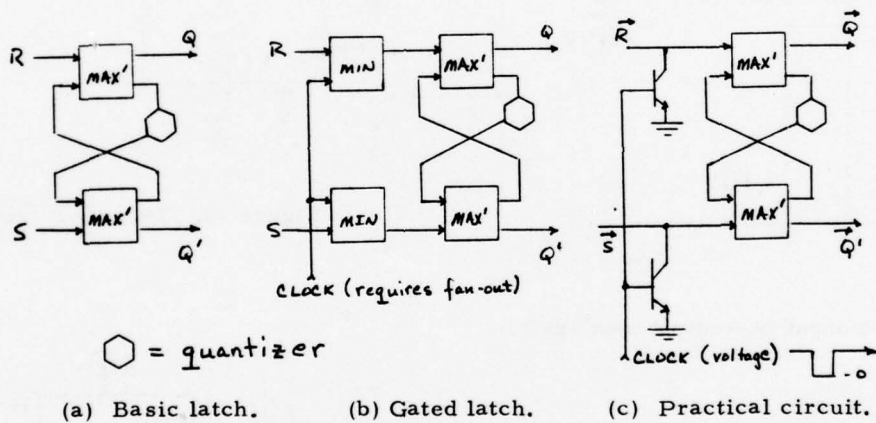


Figure 19. RS m-valued latch.

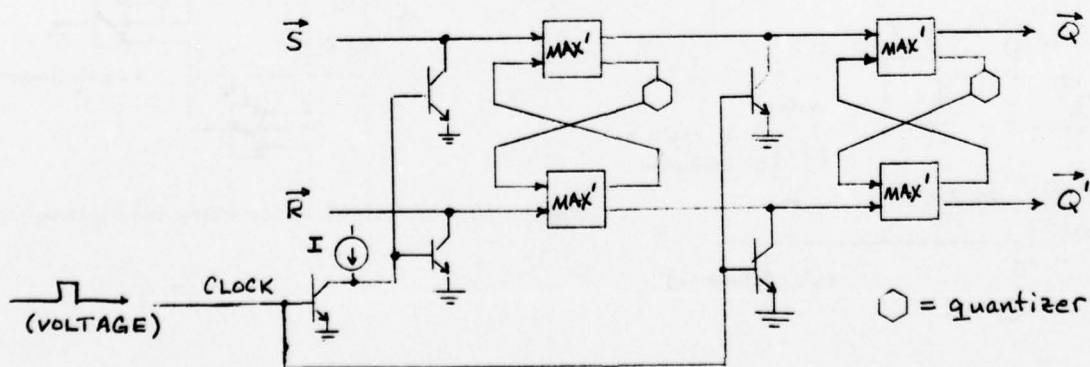


Figure 20. Master-slave RS m-flop.

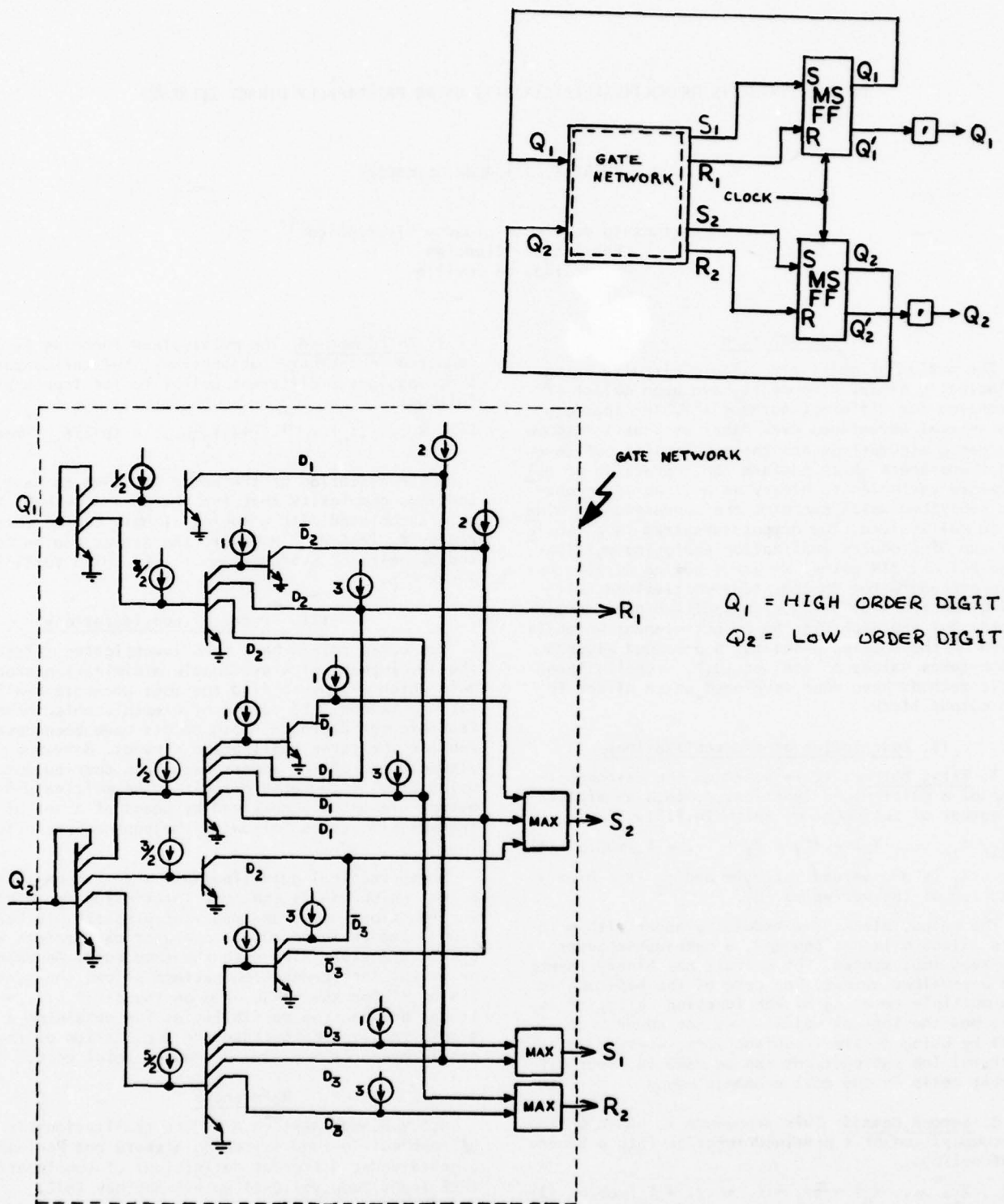


Figure 21. 4,1 quaternary-coded decimal counter.

ON THE SYNTHESIS OF MULTIVALUED CIRCUITS USING PRINCIPALLY BINARY ELEMENTS

J.L.HUERTAS, J.I.ACHA, L.MACIAS

Departamento de Electricidad y Electrónica
Facultad de Ciencias
Universidad de Sevilla

1. Introduction

The design of multivalued logic circuits using principally binary components have been object of attention for different authors ¹⁻². The approach has several advantages over other synthesis methods. The basic assumptions are the availability of some input operators which perform the conversion of multivalued variables to binary as well as of an output subsystem which performs the conversion of binary to multivalued. The output subsystem is a two level sum-of-products realization employing multivalued MAX and MIN gates. We are studying an alternative procedure for implementing multivalued switching functions by using principally binary components. Our approach uses an output element which is based on the modulus p sum for a p -valued algebra which takes values on the set $\{0,1,\dots,p-1\}$. Three basic methods have been developed which differ in the output block.

II. Description of the realizations

1. First Method: It is based on the decomposition of a multivalued function, F , into an arbitrary number of subfunctions which fulfill:

$$F(X_1, X_2, \dots, X_n) = (f_1 + f_2 + \dots + f_m) \bmod p \quad (1)$$

where X_i is a p -valued variable and f_j is a binary function of the variables X_i .

The output block is a modulus p adder with m inputs. Since m is not bounded, a cascable adder has been implemented. The circuit has binary inputs and a p -valued output. The core of the approach is the multiple covering of the function cells. If any cell has the logical value i , we can cover that cell by using $(k+1)p-i$ subfunctions, where k is arbitrary. The subfunctions can be used to cover different cells in the most economic way.

2. Second Method: This procedure is based on the decomposition of a p -valued function into p binary subfunctions:

$$F(X_1, X_2, \dots, X_n) = (f_1 + f_2 + \dots + f_p) \bmod p \quad (2)$$

This method requires an adder with a fixed number of inputs which is equal to the number of logical levels. As a counterpart, the freedom in the election of f_i is not as high as in the first approach. Now, a function cell with value i must be covered by $p-i$ subfunctions. Clearly, this particularization of the first realization when $k = 0$.

3. Third Method: The multivalued function is decomposed in p -binary subfunctions, but the output block assigns a different weight to its inputs as follows:

$$F(X_1, X_2, \dots, X_n) = [1.f_1 + 2.f_2 + \dots + (p-1)f_{p-1}] \bmod p \quad (3)$$

The realization of the adder is shown to have the same complexity that for the second method. Any cell associated with a logical i only can be covered by f_i if $i \neq 0$. However, the zeroes can be covered by pairs of subfunctions f_j, f_k which fulfill $j + k = p$.

III. Practical considerations

Two basic points have been investigated. First, the development of a systematic minimization procedure which allows to find the most adequate realization. Second, the design of elements which makes feasible the approach. Both points have been covered for the three realization schemes. Hardware implementation of the adders have been carried out by using C-MOS integrated circuits. The multivalued output element was realized by means of a set of transmission gates following the approach reported in ³.

Some practical guidelines allow to recognize a priori which one is the most interesting for a given function. Also, an important property has been found. The designed output operator can perform any cyclic transformation with the same cost. An exchange in the interconnection pattern allows the synthesis of Z^e for any value of e on the set $\{0,1,\dots,p-1\}$. It has broadened the possibilities for obtaining a simple realization because the application of the cyclic operator may give a reduced total cost.

References

- ¹ SU, S.Y.H. and SARRIS, A.A.: "The realization of multivalued switching algebra and Boolean algebra under different definitions of complement". IEEE Trans. Comp. Vol. C-21, pp. 479-485, May 1972.
- ² BIRK, J.E. and FARMER, D.E.: "An algebraic method for designing multivalued logic circuits using principally binary components". IEEE Trans. Comp. Vol. C-24, pp. 1101-1104, Nov. 1975.
- ³ HUERTAS, J.L., ACHA, J.I. and CARMONA, J.M.: "Implementation of some ternary operators with C.M.O.S integrated circuits". Electron. Lett. Vol. 12, n° 15, pp. 385-386, 1976.

A SUGGESTED APPROACH TO COMPUTER ARITHMETIC FOR DESIGNERS
OF MULTI-VALUED LOGIC PROCESSORS

D. E. Atkins

Program in Computer, Information and Control Engineering
and
Systems Engineering Laboratory
Department of Electrical and Computer Engineering
The University of Michigan
Ann Arbor, Michigan 48109

ABSTRACT

An approach to the topic of computer arithmetic is suggested which may have a particular conceptual, pedagogical, and practical appeal to the designer of multiple-valued logic processors. Computer arithmetic deals with the physical representation of finite sets of numbers and the design, analysis, and implementation of algorithms for mechanizing arithmetic operations on these sets. Finite number representation systems (FNRS) are specified by defining a set of symbols and a mapping from the elements of this symbol set to a subset of the real numbers. A formal definition of a FNRS provides a basis for a set of definitions which in turn provide the framework for the classification of a large set of number systems. Emphasis in this paper is on the following positive, fixed radix systems: unsigned, sign and magnitude, radix complement, and diminished radix complement.

We offer an annotated listing of primitive digit vector algorithms for the four common number representation systems with an arbitrary, positive integer, fixed radix. These digit vector algorithms are ones which the designer of multi-valued logic arithmetic processors will need to implement to provide general arithmetic computation.

INTRODUCTION

Although the predominance of two-valued logic has naturally led to the implementation of binary (radix 2) arithmetic in most digital computers, the theory of computer arithmetic deals with a much broader class of possibilities. At a minimum, the designer of processors with multi-valued technology will be interested in higher radix versions of standard radix polynomial systems, and the possibility exists that more novel number systems (residue, signed-digit, rational, etc.) may find practical application in a multi-valued environment.

An ongoing project within our laboratory concerns developing a unified description and classification of finite number representation systems together with a set of primitive building blocks for arithmetic design, so-called "digit-vector algorithms." One of our goals is to present the designer with a wide range of choices and a

method for assigning a figure of merit to various number systems with respect to a given implementation environment. For example, the complexity of the SUM digit vector algorithm is lower in a residue number system than in a standard radix polynomial system, however, the reverse is true for the SIGN (sign detection) digit vector algorithm. The act of defining these algorithms which simulate useful abstract arithmetic structures is what we call "arithmetic design"; traditional "logic design" comes into play only after we make the decision to represent the required digits using binary codes. As often noted by Avizienis, failure to make the distinction between "arithmetic design" and "logic design" has long plagued the literature in computer arithmetic.

The goal of this paper is to encourage collaboration between arithmetic design and the implementation of multi-valued logic. We feel that our first step must be to present definitions and notation to facilitate precise communication, and to enhance appreciation of the wide range of theoretical options available to designers. Specifically, in this paper we present definitions relating to the representation of numbers, give formal definitions of several fixed radix systems in common use, and then list a set of arithmetic building blocks, digit vector algorithms (DVAs), for performing arithmetic in these number systems.

The "approach" we are suggesting is that designers of multi-valued logic processors use the DVAs as formal definitions of the functions which they must provide in their particular circuit technology. This approach should, at a minimum, facilitate the comparison of alternate choices of number systems for given constraints, and also help to avoid the pitfalls which sometimes arise when we too quickly generalized from our radix 2 arithmetic experience. Our suggestion is to think broadly, to learn the general case, and to treat binary arithmetic only as the special case it is.

We shall use the programming language APL as a description language. Although APL is sometimes criticized for having awkward control structures and for encouraging obscurity, we feel that it is well suited for the task at hand. Knowledge of only a relatively small subset of the language is required here. References on APL include [1-9].

This work was supported by the National Science Foundation, Division of Mathematical and Computer Science under Grant No. MCS 77-03310.

REPRESENTATION OF NUMBERS

Computer arithmetic deals with the physical representation of finite sets of numbers and the design, analysis, and implementation of algorithms for mechanizing arithmetic operations on these sets. We consider numbers to be abstract entities which are defined theoretically, typically by their properties (axiomatically). Peano's Five Axioms, for example, define the properties of positive integers. In implementing computer arithmetic we assume we are given the algebra of real and complex numbers. Complex numbers (imaginary numbers) may be made to correspond to a unique pair of real numbers and therefore will not be further discussed explicitly. (Alternately we could view real numbers as being embedded in the class of complex numbers and focus on a discussion of complex numbers.) The set of real numbers include, for example, the natural numbers

$$\mathbb{N} = \{0, 1, 2, 3, \dots\},$$

the integers

$$\mathbb{Z} = \{0, \pm 1, \pm 2, \pm 3, \dots\},$$

the positive integers

$$\mathbb{P} = \{1, 2, 3, \dots\},$$

and the rational numbers

$$\mathbb{Q} = \{x \text{ such that } x = p/q \text{ and } p \in \mathbb{Z} \text{ and } q \in \mathbb{P}\}.$$

Since we are concerned with the physical mechanization of arithmetic we are restricted to representation of finite sets of numbers, i.e., to subsets of the reals. We will be particularly concerned with representation of, and operations on, the set of integers modulo N ,

$$\mathbb{Z}_N = \{0, 1, 2, \dots, (N-1), \text{ for } N \geq 2\}.$$

Rational numbers (fractions) may be treated either as an ordered pair of integers or as "scaled integers."

When we deal with arithmetic, we imply that such sets of numbers are part of an algebraic system or structure consisting of a set and one or more n -ary operations (functions) on the set. A more complete definition also frequently includes relations on the sets and distinguished elements of the set such as 0 and 1. Examples of algebraic systems include:

$\langle \mathbb{Z}, +, * \rangle$ where $+$ and $*$ are the operations of addition and multiplication on the set of integers;

$\langle \mathbb{R}, +, * \rangle$, where $+$ and $*$ are addition and multiplication on the set of reals; and

$\langle \mathbb{Z}_N, +_N \rangle$, where $+_N$ is addition module N .

To repeat then, computer arithmetic deals with the representation of finite sets of numbers which are typically subsets of the sets described above, and with the mechanization of well-known n -ary

(usually unary and binary) operations on these finite sets.

In this section we will consider the representation of numbers by symbols which are variously referred to in the literature as 1) symbol strings, 2) n -tuples, 3) code words, or 4) digit vectors. All of these terms offer some expressive advantage; "symbol strings" relates to language and data structure ideas; " n -tuple" is a standard term in discrete mathematics; "code word" conveys the well-known idea of encoding information; and "digit vector" implies a connection with vector notation and vector languages such as APL. We will reserve the right to use all three, however, our preference will be "digit vector."

We will define finite sets of symbols which can be physically represented and also operations on these sets which "simulate" well-known algebraic structures such as mentioned above. This notion of a "simulation" may be described in terms of the formal idea of homomorphism and all of the various specific subvarieties such as an isomorphism.

It is also intuitively useful to think about (finite precision) computer arithmetic as an approximation of arithmetic on the reals. The fact that it is an approximation gives rise to need for numerical analysis. Only finite precision arithmetic is mechanized on a digital computer. This is probably the most important characteristic of computer arithmetic: direct consequences of finitude include overflow, underflow, scaling, and the use of complement representation of negative quantities. In selecting a number representation system, the designer of a computer arithmetic system must keep in mind the architectural realities of time and hardware efficiency, and the numeric reality of providing an adequate approximation of real arithmetic.

Definitions

Finite number representation systems ("number systems") are usually specified by defining a set of symbols, \mathbb{A} , and a mapping from the elements of this symbol set to a subset of the set of real numbers, $F: \mathbb{A} \rightarrow \mathbb{R}$.

The elements of the symbol set, \mathbb{A} , usually have the form of a finite N -tuple which we shall call a "digit vector." In other words, a digit vector X is an element of the set \mathbb{A} where

$$\mathbb{A} = \mathbb{D}_1 \times \mathbb{D}_2 \times \dots \times \mathbb{D}_N$$

and \mathbb{D}_i is the set of allowable digit values for the i th component and \times denotes the Cartesian product.

Following the suggestions of Budkowski [10] we now introduce the following definitions:

1) The function $F: \mathbb{A} \rightarrow \mathbb{R}$ is a total function if F is defined for all elements of \mathbb{A} (all digit vectors in \mathbb{A}).

2) The function $F: \mathbb{A} \rightarrow \mathbb{R}$ is a partial function if F is not defined for all elements of \mathbb{A} . In this

case the subset of A for which F is defined is denoted AF and is called the set of "legal digit vectors." Note that in this case $AF \subseteq A$.

3) Since $AF \subseteq A$ is a finite set, the mapping $F: AF \rightarrow R$ is into R , the infinite set of all reals. The subset $RF \subseteq R$ which is the image of AF under F is called the set of representable numbers or the interpretation set.

The elements of a digit set, DI , are integers and will be taken as defined in this presentation. Note that in practice the elements of a digit set might be denoted using other than standard decimal notation. For example, in the hexadecimal system the digit values are usually taken to be elements of the set $\{0, 1, \dots, 9, A, B, C, D, E, F\}$. For additional generality numerical meanings can be associated with digit symbols by definition of a one-to-one function from symbols to numbers. Formally then,

Definition 1 A finite number representation system (abbreviated FNRS) is a triple

$$FNRS = \langle A, AF, F \rangle$$

where

A is a finite, nonempty set of digit vectors, $AF \subseteq A$ is the set of all legal digit vectors, and F is a function which maps AF into R (the set of real numbers).

In this paper we concentrate on representation of integers, i.e., RF will be a finite set of integers. Having developed integer representation, we may treat representation of rational numbers (fractions) by "scaling" integers.

Definition 2 A FNRS is said to be partial if F is a partial function in A , that is if $AF \subset A$.

Definition 3 A FNRS is said to be total if F is a total function in A , that is if $AF = A$.

Definition 4 A FNRS is said to be redundant if for $X \in AF$, $F(X)$ is a many-to-one function. In this case at least one element of the set of representable numbers (RF) has more than one representation.

Definition 5 A FNRS is said to be nonredundant if for $X \in AF$, $F(X)$ is a one-to-one function.

Although an initial reaction may be that a redundant FNRS would be a disadvantage, in mechanizing machine arithmetic redundancy may offer significant advantage. A discussion of this topic is beyond the scope of this paper but may well be of practical significance in the realization of arithmetic using multi-valued logic. For a brief overview of the role of redundancy in computer arithmetic see [11].

Definition 6 A FNRS is weighed if F is defined by the function

$$Q = (\sum X[I] \times W[I]) + C$$

where

$Q \in R$,
 $N = \rho X$, the length of digit vector X ,
 $X[I]$ is the I th element of a digit vector $X \in AF$,
 $W[I]$ is the I th element of a weight vector with $W[I] \in R$,
 and C is a constant.

In APL, the above expression can be written

$$Q \leftarrow (+/X \times W) + C$$

where $\rho X = \rho W$. The expression $+/X \times W$ is commonly called an "inner product."

An important special case of weighed FNRS are those in which the weight vector, W , is obtained from a so-called radix vector,

$$B = B[1], B[1], B[2], \dots, B[N].$$

Usually $B[I]$ is an element of Z , the set of integers (negative radices are also extensively discussed in the literature). The prospect of $B[I]$ being a complex number has also been proposed.

Definition 7 A FNRS is a weighed radix system if it is a weighed system (Def. 6) in which the elements of the weight vector W are defined as follows:

$$W[N] = 1,$$

$$W[I] = W[I+1] \times B[I+1] \text{ for } I = N-1, N-2, \dots, 1.$$

where $B[I] \in Z$ is an element of a radix vector.

Note that in our choice of the definition for W we are continuing to restrict ourselves to the representation of integers.

Definition 8 A weighed FNRS is a fixed radix (or base) system if all elements of B are the same, i.e., if $B[I] = B[J]$ for $1 \leq I, J \leq N$.

Definition 9 A weighed FNRS is a mixed radix (or base) system if all elements of B are not the same, i.e., if there exists some $I \neq J$ ($1 \leq I, J \leq N$) such that $B[I] \neq B[J]$.

Fixed radix number systems are the most commonly used. In this case, F , the function between symbols and interpretation, may be specified as a polynomial ("radix polynomial") with the digit vector comprising the coefficients. A very interesting treatment of fixed radix FNRS, based upon the ring of polynomials over the integers may be found in [12]. Radix polynomial FNRS are also called "polyadic" FNRS.

The computer language APL includes a primitive operator which mechanizes the mapping from digit vectors to integers for the radix number systems. If we represent the elements of the set of representable numbers ("the interpretation set") using standard sign and magnitude decimal notation, then

$$Q \leftarrow B \downarrow X$$

produces an element $Q \cdot B \cdot X$ where B is a radix vector and X is a digit vector. This APL operator is called DECODE. For example if

$B \leftarrow 2, 2, 2, 2$
 $X \leftarrow 1, 1, 1, 0$

then $Q \cdot B \cdot X$ is the decimal equivalent of the binary digit vector 1, 1, 1, 0. If B is 30, 24, 60, 60 and X is 5, 7, 15, 37 then $Q \cdot B \cdot X$ is the number of seconds in 5 days, 7 hours, 15 minutes, and 37 seconds.

Let us return now to a discussion of digit sets, i.e., the sets from which digit vector elements are taken. Recall that A , the set of digit vectors, was defined by

$$A = D_1 \times D_2 \times \dots \times D_N$$

where D_I ($I = 1, 2, \dots, N$) are sets of digits.

Definition 10 The canonical (or standard) digit set for the I th element of a digit vector in a radix FNRS is the set

$$D_I = \{0, 1, 2, \dots, (B[I]-1)\}$$

where $B[I]$ is the I th element of the radix vector. Note that in this case $|D_I| = B[I]$.

We will make use of other than canonical digit sets, for example, symmetric digit sets, defined as follows:

Definition 11 A symmetric digit set with respect to the positive integer K is the set

$$Z_{oK} = \{-K, -(K-1), \dots, 0, 1, \dots, (K-1), K\}.$$

Definition 12 A fixed radix FNRS with $B[I] \geq 2$ for all $1 \leq I \leq N$ with canonical digit sets is called a conventional FNRS.

EXAMPLES OF COMMONLY USED FINITE NUMBER REPRESENTATION SYSTEMS

The above definitions provide a framework for describing a large number of FNRS including residue, negative radix and signed-digit. Within the constraints of this paper, however, we will only review the definition of the generalized, positive radix versions of four commonly used FNRS: 1) Conventional, radix B , unsigned (magnitude only); 2) Conventional, radix B , sign and magnitude; 3) Conventional, radix B , radix complement; and 4) Conventional, radix B , diminished radix complement.

The definition of these FNRS are presented in Figures 2-5 using symbols defined in Figure 1. Each figure describes the symbol set, the interpretation set, and the symbol to interpretation (SI) function in the form of an APL function. These APL functions, generalizations of the APL decode (\downarrow) operator, have digit-vectors as arguments and produce a sign and magnitude, decimal representation of the corresponding element of the interpretation set. We'll use familiar notation to denote a particular element of the interpretation set.

PRIMITIVE OPERATIONS ON COMMON FINITE NUMBER REPRESENTATION SYSTEMS (DIGIT VECTOR ALGORITHMS)

To this point we have concentrated on the representation of finite sets of numbers using symbol sets consisting of digit vectors. We now present fundamental unary and binary operations on symbol sets which, together with isomorphic SI mappings such as described in the previous section, will enable us to simulate useful algebraic structures. Examples of the operations or "digit vector algorithms" include sum, difference, inverse, range extension, and range contraction. Note that in defining these algorithms we are assuming that we are given standard integer arithmetic on the individual digits.

In the remainder of this paper we describe primitive digit vector algorithms which the designer of a multi-valued logic processor must be prepared to implement. The proposed set is motivated by Avizienis [13]. Space does not permit a discussion of each algorithm, however, as an example of what might be done, we include a discussion of the SUM and CARRY DVAs.

SUM and CARRY Digit Vector Algorithms

The digit vector algorithm, SUM, corresponds to what we commonly call addition. Given an SI mapping F , we need to define SUM such that

$$F(X \text{ SUM } Y) = F(X) + F(Y)$$

where $X, Y \in A$.

For number systems (fixed or mixed base) with all elements of the radix vector, $B \geq 2$, and canonical digit set, the following digit vector operation applies:

$$S \leftarrow B \cdot X + Y + C$$

where

B is the radix vector;
 X and Y are the digit vector operands with $pX = pY$,
 C is the carry vector (to be defined),
and S is the sum vector.

The sum digit for a given position of an adder, say the I th position, is sometimes given as

$$S[I] \leftarrow (X[I] + Y[I] + C[I]) - B[I] \times C[I-1]$$

where $C[I]$ is the carry into the position, $C[I-1]$ is the carry out, and $B[I]$ is the I th element of the radix vector. This form may better correspond to intuition, namely, that the sum digit is the sum of the two operand digits and the carry in ($X[I] + Y[I] + C[I]$) minus a correction. If the sum is greater than the maximum digit we can represent ($B[I]-1$) then we subtract $B[I]$ from the I th position and add 1 in the position $I-1$. Since the weight of position $I-1$ is $W[I-1] = B[I] \times W[I]$, this subtraction of $B[I]$ in position I and addition of 1 in position $I-1$ do not change the value represented by the digit vector.

Listing of DVAs for common FNRS

We conclude by offering an annotated listing of primitive digit vector algorithms for the four common finite number representation systems we have reviewed. In exchange for the reader's willingness to read APL, he/she will find a formal description of operations which should be implemented for a favorite choice of radix, B, and choice of common number system. Similar work on less conventional but potentially practical number systems is underway and the interested reader is invited to contact the author for further information.

REFERENCES

1. Sandra Pakin, APL 360 Reference Manual, Second Edition, SRA, Chicago, 1972.
2. G. Demars, J. C. Rault, G. Ruggio, Le Langage et Les Systems APL, (in French), Masson et Cie, Paris, 1974.
3. APL 360 User's Manual, IBM Data Processing Division.
4. APL Shared Variables (APLSV) User's Guide, IBM (SH20-1460).
5. APL Language, IBM, (GC26-3847).
6. University of Michigan Computing Center Memo 363, MTS APL User's Guide.
7. L. Gilman and A. Rose, APL 360, An Interactive Approach, John Wiley and Sons, New York, 1970.
8. K. E. Iverson, A Programming Language, Wiley, New York, 1962.
9. H. Katzan, APL Programming and Computer Techniques, van Nostrand Reinhold, 1970.
10. S. Budkowski, D. Atkins, "A unified classification of finite number systems," Systems Engineering Lab. Report No. 106, ECE Dept., University of Michigan, Ann Arbor.
11. D. Atkins, "The role of redundancy in computer arithmetic," Computer, June 1975, Vol. 8, No. 6, pp. 74-76.
12. D. Matula, "Radix arithmetic: Digital algorithms for computer architecture," Chapter 9 in Applied Computation Theory: Analysis, Design, Modeling, R. Yeh, ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1976.
13. A. Avizienis, "Digital Computer arithmetic: A unified algorithmic specification," Proceedings of the Symposium on Computers and Automata, New York, 1971, Polytechnic Press, New York. Available through Wiley-Interscience.

\underline{A} = SET OF ALL DIGIT VECTORS.
 \underline{AE} = SET OF ALL LEGAL DIGIT VECTOR (THOSE FOR WHICH F IS DEFINED)
 \underline{E} = THE SET OF REAL NUMBERS.
 \underline{FE} = THE INTERPRETATION SET, I.E. THE IMAGE OF \underline{AE} UNDER F.
 B = FIXED VALUE OF ALL ELEMENTS OF THE RADIX VECTOR. $B \geq 2$.
 $\underline{ZB} \leftarrow \underline{B}$ = THE SET OF INTEGERS MODULO B.
 X = ELEMENT OF \underline{AE} , I.E. A LEGAL DIGIT VECTOR.
 Q = ELEMENT OF \underline{FE} .
 ρX = LENGTH OF DIGIT VECTOR, X.
 B^*N = B TO THE POWER N.
 $N \text{ CP } X$ = N-ARY CARTESIAN PRODUCT OF SET X, I.E. $X \times X \times \dots \times X$, N TIMES.
 $1 \downarrow X$ = THE FIRST ELEMENT OF VECTOR X.
 $1 \uparrow X$ = ALL BUT THE FIRST ELEMENT OF X.
 $\lfloor K$ = FLOOR OF K, I.E. THE GREATEST INTEGER $\leq K$.
 $\lceil K$ = CEILING OF K, I.E. THE SMALLEST INTEGER $\geq K$.

FIGURE 1. SYMBOLS USED IN DEFINING NUMBER SYSTEMS IN FIGURES 2-5

SYMBOL SET: $A \leftarrow AE \leftarrow N \text{ CP } \setminus B$

INTERPRETATION SET: $RE \leftarrow \setminus (B * N)$

SI FUNCTION: $F: AE \rightarrow RE$

$\forall Q \leftarrow B \text{ DECODEUS } X$
[1] $\text{ASI FUNCTION FOR CONVENTIONAL RADIX } B \text{ UNSIGNED FNRS}$
[2] $Q \leftarrow B \setminus X$

\forall
EXAMPLES

2 DECODEUS 0 1 1 0
6

2 DECODEUS 1 0 0 1
9

10 DECODEUS 0 3 1 9
319

5 DECODEUS 1 2 3 4
194

8 DECODEUS 3 5 7 7
1919

FIGURE 2. DEFINITION OF CONVENTIONAL, RADIX B UNSIGNED FNRS.

SYMBOL SET: $A \leftarrow AE \leftarrow S \times N \text{ CP } \setminus B$

WHERE $S \leftarrow (0, 1)$ AND N IS THE LENGTH OF THE DIGIT VECTOR
REPRESENTING THE MAGNITUDE. IN S , 0 DENOTES + AND 1 DENOTES -
NOTE THAT IF $X \in AE$ THEN $N = (\rho X) - 1$

INTERPRETATION SET: $RE \leftarrow -K, \dots, -1, 0, 1, 2, \dots, K$
WHERE $K \leftarrow (B * N) - 1$

SI FUNCTION: $F: AE \rightarrow RE$

$\forall Q \leftarrow B \text{ DECODESM } X$
[1] $\text{ASI FUNCTION FOR CONVENTIONAL RADIX } B \text{ SIGN AND MAGNITUDE FNRS}$
[2] $Q \leftarrow (-1 \setminus 1 \setminus X) \times B \setminus 1 \setminus X$

\forall
EXAMPLES

2 DECODESM 0 1 1 0
6

2 DECODESM 1 0 0 1
-1

10 DECODESM 0 3 1 9
319

5 DECODESM 1 2 3 4
-69

8 DECODESM 0 5 7 7
383

FIGURE 3. DEFINITION OF CONVENTIONAL, RADIX B SIGN AND MAGNITUDE FNRS.

SYMBOL SET: $A \leftarrow AE \leftarrow N \text{ CP } \setminus B$

INTERPRETATION SET:

RADIX B, EVEN: $RE \leftarrow -K, \dots, -1, 0, 1, \dots, (K-1)$
 WHERE $K \leftarrow (B \cdot N) \div 2$

RADIX B, ODD: $RE \leftarrow -K_1, \dots, -1, 0, 1, \dots, K_2$
 WHERE $K_1 \leftarrow (\lfloor B \div 2 \rfloor) \times B \cdot (N-1)$ AND $K_2 \leftarrow ((B \cdot (N-1)) \times (\lceil B \div 2 \rceil)) - 1$

SI FUNCTION: $F: AE \rightarrow RE$

- $\forall Q \leftarrow B \text{ DECODERC } X$
- [1] A SI FUNCTION FOR CONVENTIONAL RADIX B, RADIX COMPLEMENT FNRS
 - [2] $Q \leftarrow (B \setminus X) - ((1 \setminus X) \geq B \div 2) \times B \cdot \rho X$
 - [3] A NOTE THAT FOR THE SPECIAL CASE OF $B=2$ THE FOLLOWING APPLIES:
 - [4] $AQ \leftarrow 2 \setminus (-1 \setminus X), 1 \setminus X$
 - [5] A I.E. WE TREAT THE SIGN INDICATOR DIGIT AS HAVING NEGATIVE WEIGHT.

EXAMPLES

$\frac{2}{7} \text{ DECODERC } 1 \ 0 \ 0 \ 1$

$\frac{10}{319} \text{ DECODERC } 0 \ 3 \ 1 \ 9$

$\frac{5}{181} \text{ DECODERC } 3 \ 2 \ 3 \ 4$

$\frac{8}{1919} \text{ DECODERC } 3 \ 5 \ 7 \ 7$

FIGURE 4. DEFINITION OF CONVENTIONAL, RADIX B, RADIX COMPLEMENT FNRS.

SYMBOL SET: $A \leftarrow AE \leftarrow N \text{ CP } \setminus B$

INTERPRETATION SET:

RADIX B, EVEN: $RE \leftarrow -K, \dots, -1, -0, +0, 1, \dots, K$
 WHERE $K \leftarrow ((B \cdot N) \div 2) - 1$

RADIX B, ODD: $RE \leftarrow -K_1, \dots, -1, -0, +0, 1, \dots, K_2$
 WHERE $K_1 \leftarrow ((\lfloor B \div 2 \rfloor) \times B \cdot (N-1)) - 1$ AND $K_2 \leftarrow ((B \cdot (N-1)) \times \lceil B \div 2 \rceil) - 1$

SI FUNCTION: $F: AE \rightarrow RE$

- $\forall Q \leftarrow B \text{ DECODEDC } X$
- [1] A SI FUNCTION FOR CONVENTIONAL RADIX B, DIMINISHED
 - [2] A RADIX COMPLEMENT FNRS
 - [3] $Q \leftarrow (B \setminus X) - ((1 \setminus X) \geq B \div 2) \times (B \cdot \rho X) - 1$

EXAMPLES

$\frac{2}{6} \text{ DECODEDC } 1 \ 0 \ 0 \ 1$

$\frac{10}{319} \text{ DECODEDC } 0 \ 3 \ 1 \ 9$

$\frac{5}{180} \text{ DECODEDC } 3 \ 2 \ 3 \ 4$

$\frac{8}{1919} \text{ DECODEDC } 3 \ 5 \ 7 \ 7$

FIGURE 5. DEFINITION OF CONVENTIONAL, RADIX B, DIMINISHED RADIX COMPLEMENT FNRS.

DIGIT VECTOR ALGORITHMS
COMMON FINITE NUMBER REPRESENTATION SYSTEMS
FIXED-RADIX
JANUARY 1978, REVISION 4

THESE DIGIT VECTOR ALGORITHMS ARE DEFINED FOR THE SPECIAL CASE OF FIXED-RADIX FNRS. A SCALAR GLOBAL VARIABLE B, WHICH MUST BE ≥ 2 , IS THE RADIX.

ABBREVIATIONS FOR DIGIT VECTOR ALGORITHM FAMILY NAME:

SUM, CARRY
DIF (DIFFERENCE), BORROW
INV (ADDITIVE INVERSE)
SGN (SIGN DETECTION), EQZ (EQUAL ZERO)
RFX (RANGE EXTENSION), RCN (RANGE CONTRACTION)
SDN (SCALE DOWN), SUP (SCALE UP)
PRD (PRODUCT)

ABBREVIATIONS FOR NUMBER SYSTEM NAME:

US - CONVENTIONAL, UNSIGNED
RC - RADIX COMPLEMENT
DRC - DIMINISHED RADIX COMPLEMENT
SM - CONVENTIONAL SIGN AND MAGNITUDE

ABBREVIATIONS FOR SINGULARITIES:

OVF - OVERFLOW
AN - ANOMALY
TR - TRUNCATIONS

THE ABOVE ARE FOLLOWED BY DVA FAMILY NAME AND NUMBER SYSTEM NAME, FOR EXAMPLE, OVFSUMUS.

ASSUMPTIONS:

1. FOF DYADIC FUNCTIONS $\rho X = \rho Y$.
2. THE RADIX (B) IS ALWAYS A SCALAR. THE RADIX VECTOR FOR A DIGIT VECTOR X IS CONCEPTUALIZED TO BE $(\rho X)\rho B$. B MUST BE ≥ 2 BUT MAY BE ODD OR EVEN.
3. FOF ALL VARIATIONS OF SUM AND DIF, THE CARRY IN (CIN) AND BORROW IN (BIN) WILL BE DEFINED EXTERNAL TO THE APL FUNCTION.
4. IN RCN, SDN, AND SUP THE VALUE OF M MUST BE $\leq \rho X$ FOR US, RC, AND DRC; M MUST BE $< \rho X$ FOR DRC.

```

▽ S+X SUMUS Y
[1]  A SUM DVA FOR CONVENTIONAL UNSIGNED FNRS.
[2]  A CIN AND A FIXED RADIX B ARE DEFINED EXTERNALLY
[3]  S←B|X+Y+X CARRYUS Y
[4]  OVFSUMUS←COUT
▽

▽ C+X CARRYUS Y;I;ORG
[1]  A CARRY DVA FOR CONVENTIONAL UNSIGNED FNRS.
[2]  A RADIX B AND CARRY IN CIN ARE EXTERNALLY DEFINED.
[3]  ORG←1
[4]  I←(pX)+ORG-1
[5]  C←(((pX)-1)p0),CIN
[6]  LOOPPCPR:←(ORG=I)/LCOUT
[7]  C[I-1]←B≤X[I]+Y[I]+C[I]
[8]  ←(ORG<I+I-1)/LOOPPCPR
[9]  LCOUT:COUT←B≤X[ORG]+Y[ORG]+C[ORG]
▽

▽ S+X SUMSM Y;SIGNX;SIGNY;SIGNS;MAGX;MAGY;MAGS
[1]  A SUM DVA FOR CONVENTIONAL SIGN AND MAGNITUDE FNRS.
[2]  SIGNX←SGNSM X
[3]  SIGNY←SGNSM Y
[4]  MAGX←1+X
[5]  MAGY←1+Y
[6]  ←(SIGNX=SIGNY)/LEQSIGN
[7]  ASIGNX ≠ SIGNY
[8]  BIN←0
[9]  MAGS←MAGX DIFUS MAGY
[10]  SIGNS←SIGNX≠BOUT
[11]  ←(∼BOUT)/LUEQSIGN
[12]  BIN←0
[13]  MAGS←MAGY DIFUS MAGX
[14]  LUEQSIGN:S←SIGNS,MAGS
[15]  OVFSUMSM←0
[16]  ←LZEROCHECK
[17]  CIN←0
[18]  LEQSIGN:S←SIGNX,MAGX SUMUS MAGY
[19]  OVFSUMSM←COUT
[20]  A FORCE ONLY + ZERO
[21]  LZEROCHECK:←(0=EQZSM S)/0
[22]  S[11]←0
▽

▽ S+X SUMRC Y
[1]  A SUM DVA FOR RADIX COMPLEMENT FNRS.
[2]  S+X SUMUS Y
[3]  OVFSUMRC1←((SGNRC X)≠SGNRC Y)^(SGNRC X)≠SGNRC S
[4]  OVFSUMRC2←((SGNRC X)≠SGNRC Y)∧OVFSUMUS∧∼SGNRC X
[5]  OVFSUMRC←OVFSUMRC1∨OVFSUMRC2
▽

▽ S+X SUMDRC Y
[1]  A SUM DVA FOR DIMINISHED RADIX COMPLEMENT FNRS.
[2]  S+X SUMUS Y
[3]  CIN←COUT
[4]  S+X SUMUS Y
[5]  OVFSUMDRC1←((SGNDRC X)≠SGNDRC Y)^(SGNDRC X)≠SGNDRC S
[6]  OVFSUMDRC2←((SGNDRC X)≠SGNDRC Y)∧OVFSUMUS∧∼SGNDRC X
[7]  OVFSUMDRC←OVFSUMDRC1∨OVFSUMDRC2
▽

```

```

V D+X DIFUS Y
[1] A DIFFERENCE DVA FOR CONVENTIONAL, UNSIGNED FNRS.
[2] D+B!(X-Y)-X BORROWUS Y
[3] OVFDIFUS←BOUT
V

V BRW←X BORROWUS Y; I; ORG
[1] A BORROW DVA FOR CONVENTIONAL UNSIGNED FNRS
[2] ORG←1
[3] I←(ρX)+ORG-1
[4] BRW←(((ρX)-1)ρ0), BIN
[5] LOOPBPR:→(ORG=I)/LBOUT
[6] BRW[I-1]←((X[I]-Y[I])-BRW[I])<0
[7] →(ORG<I+I-1)/LOOPBPR
[8] LBOUT:BOUT←((X[ORG]-Y[ORG])-BRW[ORG])<0
V

V D+X DIFSM Y; SIGNX; SIGNY; SIGND; MAGX; MAGY; MAGD
[1] A DIFFERENCE DVA FOR CONVENTIONAL SIGNED MAGNITUDE FNRS.
[2] SIGNX←SGNSM X
[3] SIGNY←SGNSM Y
[4] MAGX←1+X
[5] MAGY←1+Y
[6] →(SIGNX≠SIGNY)/LUNEQSIGN
[7] A *** SIGN(X) = SIGN(Y). ***
[8] BIN←0
[9] MAGD←MAGX DIFUS MAGY
[10] SIGND←SIGNX*BOUT
[11] →(~BOUT)/LCOMBINE
[12] BIN←0
[13] MAGD←MAGY DIFUS MAGX
[14] LCOMBINE:D←SIGND, MAGD
[15] OVFDIFSM←0
[16] →LEQZCHECK
[17] LUNEQSIGN:CIN←0
[18] D←SIGNX, MAGX SUMUS MAGY
[19] OVFDIFSM←COUT
[20] A *** FORCE ONLY POSITIVE ZERO ***
[21] LEQZCHECK:→(0=EQZSM D)/0
[22] D[1]←0
V

V D+X DIFRC Y
[1] A DIFFERENCE DVA FOR RADIX COMPLEMENT FNRS.
[2] D+X DIFUS Y
[3] OVFDIFRC1←((SGNRC X)≠SGNRC Y)^(SGNRC X)≠SGNRC D
[4] OVFDIFRC2←((SGNRC X)≠SGNRC Y)∧OVFDIFUS∧~SGNRC X
[5] OVFDIFRC←OVFDIFRC1∨OVFDIFRC2
V

V D+X DIFDRC Y
[1] A DIFFERENCE DVA FOR DIMINISHED RADIX COMPLEMENT FNRS.
[2] D+X DIFUS Y
[3] BIN←BOUT
[4] D+X DIFUS Y
[5] OVFDIFDRC1←((SGNDRC X)≠SGNDRC Y)^(SGNDRC X)≠SGNDRC D
[6] OVFDIFDRC2←((SGNDRC X)≠SGNDRC Y)∧OVFDIFUS∧~SGNDRC X
[7] OVFDIFDRC←OVFDIFDRC1∨OVFDIFDRC2
V

```



```

      ▽ Y←INVSM X
[1]  A ADDITIVE INVERSE DVA FOR CONVENTIONAL
[2]  A SIGN AND MAGNITUDE FNRS
[3]  Y←(1+X),1+X
[4]  A GENERATE AN ANOMALY SIGNAL TO INDICATE
[5]  A THAT A NEGATIVE ZERO IS GENERATED
[6]  ANINVSM←0=+/X
      ▽

      ▽ Y←INVRC X
[1]  A ADDITIVE INVERSE DVA FOR
[2]  A RADIX COMPLEMENT FNRS
[3]  CIN←1
[4]  Y←((pX)p0)SUMRC((pX)pB-1)-X
[5]  A OVERFLOW WILL OCCUR IF SIGN(X) ≠ SIGN(Y)
[6]  A AND Y ≠ 0.
[7]  OVFINVRC←((SGNRC X)≠SGNRC Y)∧~EQZRC Y
      ▽

      ▽ Y←INVDRC X
[1]  A ADDITIVE INVERSE DVA FOR
[2]  A DIMINISHED RADIX COMPLMENT FNRS
[3]  Y←((pX)pB-1)-X
[4]  A GENERATE AN ANOMALY SIGNAL TO INDICATE
[5]  A THAT A NEGATIVE ZERO IS GENERATED
[6]  ANINVDRC←0=+/X
[7]  A OVERFLOW WILL OCCUR IF SIGN(X) ≠ SIGN(Y).
[8]  OVFINVDRC←(SGNDRC X)≠SGNDRC Y
      ▽

      ▽ SIGN←SGNSM X
[1]  A SIGN TEST FOR SIGN AND MAGNITUDE
[2]  SIGN←1+X
      ▽

      ▽ SIGN←SGNRC X
[1]  A SIGN OF X IN RADIX COMPLEMENT FNRS.
[2]  A SIGN=0 IF X IS POSITIVE.
[3]  A SIGN=1 IF X IS NEGATIVE.
[4]  SIGN←(1+X)≥B÷2
      ▽

      ▽ SIGN←SGNDRC X
[1]  A SIGN OF X IN DIMINISHED RADIX COMPLEMENT FNRS.
[2]  A SIGN=0 IF X IS POSITIVE.
[3]  A SIGN=1 IF X IS NEGATIVE.
[4]  SIGN←(1+X)≥B÷2
      ▽

      ▽ Z←EQZUS X
[1]  A TEST OF ZERO FOR UNSIGNED FNRS
[2]  Z←1/X=0
      ▽

      ▽ TEST←EQZSM X
[1]  A EQUAL ZERO TEST FOR SIGN AND MAGNITUDE FNRS
[2]  TEST←1/1+X=0
      ▽

```

▽ TEST←EQZRC X
 [1] A EQUAL ZERO TEST FOR RADIX COMPLEMENT FNRS
 [2] TEST← $\wedge/X=0$
 ▽

▽ TEST←EQZDRC X
 [1] A EQUAL ZERO TEST FOR DIMINISHED RADIX COMPLEMENT FNRS
 [2] TEST← $(\wedge/X=0) \vee (\wedge/X=B-1)$
 ▽

▽ Z←M REXUS X
 [1] A CONVENTIONAL UNSIGNED FNRS RANGE EXTENSION
 [2] A X = DV TO BE EXTENDED
 [3] A M = NUMBER OF POSITIONS TO EXTEND (SCALAR)
 [4] Z←(M_{D0}),X
 ▽

▽ Z←M REXSM X
 [1] A CONVENTIONAL SIGNED MAGNITUDE FNRS RANGE EXTENSION
 [2] A OPERANDS ARE DEFINED AS IN REXUS
 [3] Z←(1+X), (M_{D0}), 1+X
 ▽

▽ Z←M REXRC X
 [1] A RADIX COMPLEMENT FNRS RANGE EXTENSION
 [2] A OPERANDS ARE DEFINED AS IN REXUS
 [3] Z←((M_{D0}-1)×SGNRC X),X
 ▽

▽ Z←M REXDRC X
 [1] A DIMINISHED RADIX COMPLEMENT FNRS RANGE EXTENSION
 [2] A OPERANDS ARE DEFINED AS IN REXUS
 [3] Z←M REXRC X
 ▽

▽ Z←M RCNUS X
 [1] A CONVENTIONAL UNSIGNED FNRS RANGE CONTRACTION
 [2] A X = DV TO BE CONTRACTED
 [3] A M = NUMBER OF POSITIONS TO CONTRACT (SCALAR)
 [4] Z←M+X
 [5] OVERFCNUS← $1 \neq \wedge / 0 = M+X$
 ▽

▽ Z←M RCNSM X
 [1] A CONVENTIONAL SIGN AND MAGNITUDE FNRS RANGE CONTRACTION
 [2] A OPERANDS ARE DEFINED AS IN RCNUS
 [3] Z←(1+X), M+1+X
 [4] OVERFCNSM← $1 \neq \wedge / 0 = M+1+X$
 ▽

▽ Z←M RCNRC X
 [1] A RADIX COMPLEMENT FNRS RANGE CONTRACTION
 [2] A OPERANDS ARE DEFINED AS IN RCNUS
 [3] Z←M+X
 [4] OVERFCNRC← $((SGNRC Z) \wedge 1 \neq \wedge / (B-1) = M+X) \vee (\sim SGNRC Z) \wedge \sim EQZRC M+X$
 ▽

▽ Z←M RCNDR C X
 [1] R DIMINISHED RADIX COMPLEMENT FNRS RANGE CONTRACTION
 [2] R OPERANDS ARE DEFINED AS IN RCNUS
 [3] Z←M RCNRC X
 [4] OVFR CNDR C←OVFR CNRC
 ▽

▽ Z←M SDNUS X
 [1] R CONVENTIONAL UNSIGNED FNRS SCALE DOWN
 [2] R X = DV TO BE SCALED DOWN
 [3] R M = NUMBER OF PLACES TO SCALE
 [4] Z←(Mp0), (-M)↑X
 [5] TRSDNUS←1≠^/0=(-M)↑X
 ▽

▽ Z←M SDNSM X
 [1] R CONVENTIONAL SIGNED MAGNITUDE FNRS SCALE DOWN
 [2] R OPERANDS ARE DEFINED AS IN SDNUS
 [3] Z←(1↑X), (Mp0), (-M)↑1↑X
 [4] TRSDNSM←1≠^/0=(-M)↑1↑X
 ▽

▽ Z←M SDNRC X
 [1] R RADIX COMPLEMENT FNRS SCALE DOWN
 [2] R OPERANDS ARE DEFINED AS IN SDNUS
 [3] Z←(Mp(B-1)×SGNRC X), (-M)↑X
 [4] TRSDNRC←1≠^/0=(-M)↑X
 ▽

▽ Z←M SDNDR C X; SIGN
 [1] R DIMINISHED RADIX COMPLEMENT FNRS SCALE DOWN
 [2] R OPERANDS ARE DEFINED AS IN SDNUS
 [3] Z←(MpSIGN+(B-1)×SGNDR C X), (-M)↑X
 [4] TRSDNDR C←1≠^/SIGN=(-M)↑X
 ▽

▽ Z←M SUPUS X
 [1] R CONVENTIONAL UNSIGNED FNRS SCALE UP
 [2] R X = DV TO BE SCALED
 [3] R M = NUMBER OF PLACES TO SCALE
 [4] RM<pX
 [5] Z←(M↑X), Mp0
 [6] OVFSUPUS←1≠^/0=M↑X
 ▽

▽ Z←M SUPSM X
 [1] R CONVENTIONAL SIGN AND MAGNITUDE FNRS SCALE UP
 [2] R X = DV TO BE SCALED. M= NUMBER OF PLACES TO BE SCALED.
 [3] RM<pX
 [4] Z←(1↑X), ((M+1)↑X), Mp0
 [5] OVFSUPSM←1≠^/0=M↑1↑X
 ▽

▽ Z←M SUPRC X
 [1] R RADIX COMPLEMENT FNRS SCALE UP
 [2] R X = DV TO BE SCALED
 [3] R M = NUMBER OF PLACES TO SCALE
 [4] Z←(M↑X), Mp0
 [5] OVFSUPRC←((SGNRC Z)↑1≠^/(B-1)=M↑X)∨(∼SGNRC Z)↑∼EQZRC M↑X
 ▽

▽ Z←M SUPDRC X
 [1] R DIMINISHED RADIX COMPLEMENT FNRS SCALE UP
 [2] R X = DV TO BE SCALED
 [3] R M = NUMBER OF PLACES TO SCALE
 [4] Z←(M+X),Mp(B-1)×SGNDRC X
 [5] OVFSUPDRC←((SGNDRC Z)∧1≠∧/(B-1)=M+X)∨(∼SGNDRC Z)∧1≠∧/0=M+X
 ▽

▽ P←M PRDUS X;PS;PC
 [1] R POSITIVE INTEGER (M≤B-1) PRODUCTS VECTOR (X) FOR
 [2] R CONVENTIONAL, RADIX B, UNSIGNED FNRS
 [3] PS←B|M×X
 [4] PC←[(M×X)÷B
 [5] CIN←0
 [6] P←(1 REXUS PS)SUMUS 1 SUPUS 1 REXUS PC
 ▽

▽ P←M PRDSM X
 [1] R POSITIVE INTEGER (M≤B-1) PRODUCTS VECTOR (X) FOR
 [2] R CONVENTIONAL, RADIX B, SIGN AND MAGNITUDE FNRS
 [3] P←(SGNSM X),M PRDUS 1 RCNUS X
 ▽

▽ P←M PRDRC X
 [1] R POSITIVE INTEGER (M≤B-1) PRODUCTS VECTOR (X) FOR
 [2] R CONVENTIONAL, RADIX B, RADIX COMPLEMENT
 [3] P←M PRDUS X
 [4] +(0=SGNRC X)/0
 [5] R CORRECTION FOR NEGATIVE NUMBERS
 [6] CIN←0
 [7] P←P SUMUS(ρX)SUPUS(ρX)REXUS B-M
 ▽

▽ P←M PRDDRC X
 [1] R POSITIVE INTEGER (M≤B-1) PRODUCTS VECTOR (X) FOR
 [2] R CONVENTIONAL, RADIX B, DIMINISHED RADIX COMPLEMENT FNRS
 [3] P←M PRDRC X
 [4] +(0=SGNDRC X)/0
 [5] R CORRECTION FOR NEGATIVE NUMBERS
 [6] CIN←0
 [7] P←P SUMUS(ρX)REXUS M-1
 ▽

SPECIAL-PURPOSE TERNARY COMPUTER FOR DIGITAL FILTERING

Tatsuo HIGUCHI and Hisamitsu HOSHI

Department of Electronic Engineering, Faculty of Engineering
Tohoku University, Aoba, Aramaki, Sendai, Japan

ABSTRACT

Real-time digital filters which have several advantages over analog filters are known as a very important digital-signal processor. This paper provides a summary of the design features of a special-purpose microprogram-controlled ternary computer suited for realizing the real-time digital filters by programming. Many advantages of the ternary number system are effectively employed for the ternary computer in hardware and software. Particular emphasis is placed on the use of the ternary adder with three inputs and the microprogram structure based on ternary logic. An example of the use of the ternary computer proposed for the real-time digital filters is presented to illustrate how microprogramming and user programming can easily be done and execution steps can substantially be reduced, unlike the corresponding binary computer.

I. INTRODUCTION

It is well known that digital filtering is a fundamental signal-processing operation of great utility in many branches of science and technology, and digital filters for on-line, real-time applications have several advantages over analog filters [1-4]. Digital-filter computational algorithms obtained by merely a suitable combination of standard mathematical operations such as addition, subtraction, multiplication and delay may be carried out by general-purpose digital computers or special-purpose hardware. For some applications, however, the formers are too expensive and the latter are inflexible. One solution to these problems is to use microprocessors [5,6].

On the other hand, it is said that a symmetric ternary-number representation which has many advantages can simplify the construction of digital systems [7-10]. The authors have proposed a ternary digital filter as a special-purpose hardware based on the symmetric ternary-number representation using digits +1, 0 and -1, and established that the ternary digital filter thus obtained is suited for carrying out digital filtering as compared with the case of the binary digital filter [11]. In the ternary digital filter as the special-purpose hardware, however, arbitrary characteristics and structures have not been obtained because of inflexibility. Therefore, it is expected

that arbitrary digital filters may be realized by programming on the special-purpose ternary computer.

The SETUN and TERNAC are known as the full scale implementations of ternary computers [12]. However, little is known as to how the advantages of the symmetrical ternary number system are applied to the digital-filter oriented ternary computer in hardware and software.

The purpose of this paper is to design and implement a special-purpose microprogram-controlled ternary computer which is suitable for digital filtering. In the computer, an adder with three inputs is used considering the many-input addition which is frequently required for digital filtering. The adder consists of only one stage of ternary full adders unlike the case of the binary adder. The feature can effectively be employed for detecting overflow in the adder which causes overflow oscillation in digital filters [13]. In multiplication, microprogram steps and execution steps needed are reduced utilizing the following advantages: the economy of digits, the unnecessary of a special rounding off technique and a sign conversion, and 3-valued branching. Moreover, many microinstruction fields are defined by one trit. This can simplify a control unit in the ternary computer because the microinstruction fields are carried out by only one output signal.

The practical implementation is realized using the ternary threshold gates as a basic building block [11] except for the ternary ROM and RAM realized based on the binary ROM and RAM. An example for the canonical realization of a 2nd-order digital filter is presented to demonstrate how microprogramming and user programming can easily be done. In this example, it is established that the required execution steps can substantially be reduced as compared with the case of the corresponding binary computer.

II. ARCHITECTURE

In this section, the architecture is considered as the machine as seen by the programmer. The machine is a special-purpose computer operating with 10-trit (ternary digit) ternary numbers corresponding to 16-bit binary numbers. A 10-trit instruction word consists of a 4-trit operator and a 6-trit operand.

Instruction Set

In general, many differences between the binary and ternary instructions will not be recognized in the machine-language level. However, the instruction set used here includes several features not found in other binary computers. The most frequently used instructions are ADD, MULTIPLE, LOAD, STORE and JUMP. In addition operation, three-input addition which is frequently required for digital filtering can be done by only one stage of the ternary full adders, so that machine-language programming becomes more elegant and simple in detecting overflow which is very important for realizing real-time digital filters. Furthermore, both addition and subtraction are programmed without considering signs. In multiplication operation, the problem of rounding off noise is an important design consideration [1-4]. In machine-language programming of the ternary computer, special rounding off considerations are unnecessary.

Address Space

By using 6 trits, one can directly address $3^6 = 729$ memory locations. The main memory is composed of 729 words, each of them having 10 trits. In general, digital filtering can be carried out without large capability of memory. Therefore, the digital filters will easily be implemented using the special-purpose computer. In the canonic form for the nth-order digital filter, the number of memory locations required is shown in Table 1.

Table 1

Items	Number of memory locations
Multiplier coefficients	$2n + 1$
Variable data	$4n + 2$
Programming area	
Multiplication	$3(2n+1)$
3-input addition	$3n$
Delay	$2n$
Total	$17n + 6$

Multiplier coefficients determine the characteristics of digital filter obtained by programming. The number of memory locations for the variable data corresponds to the number of nodes in the block diagram representing the digital filters. Under the present capability of memory, this ternary computer can implement about 42nd-order digital filters.

III. ORGANIZATION

Fig. 1 shows a block diagram of the ternary computer proposed here. The structure of the computer consists of three main units: the register and arithmetic unit, the main memory, and the control unit.

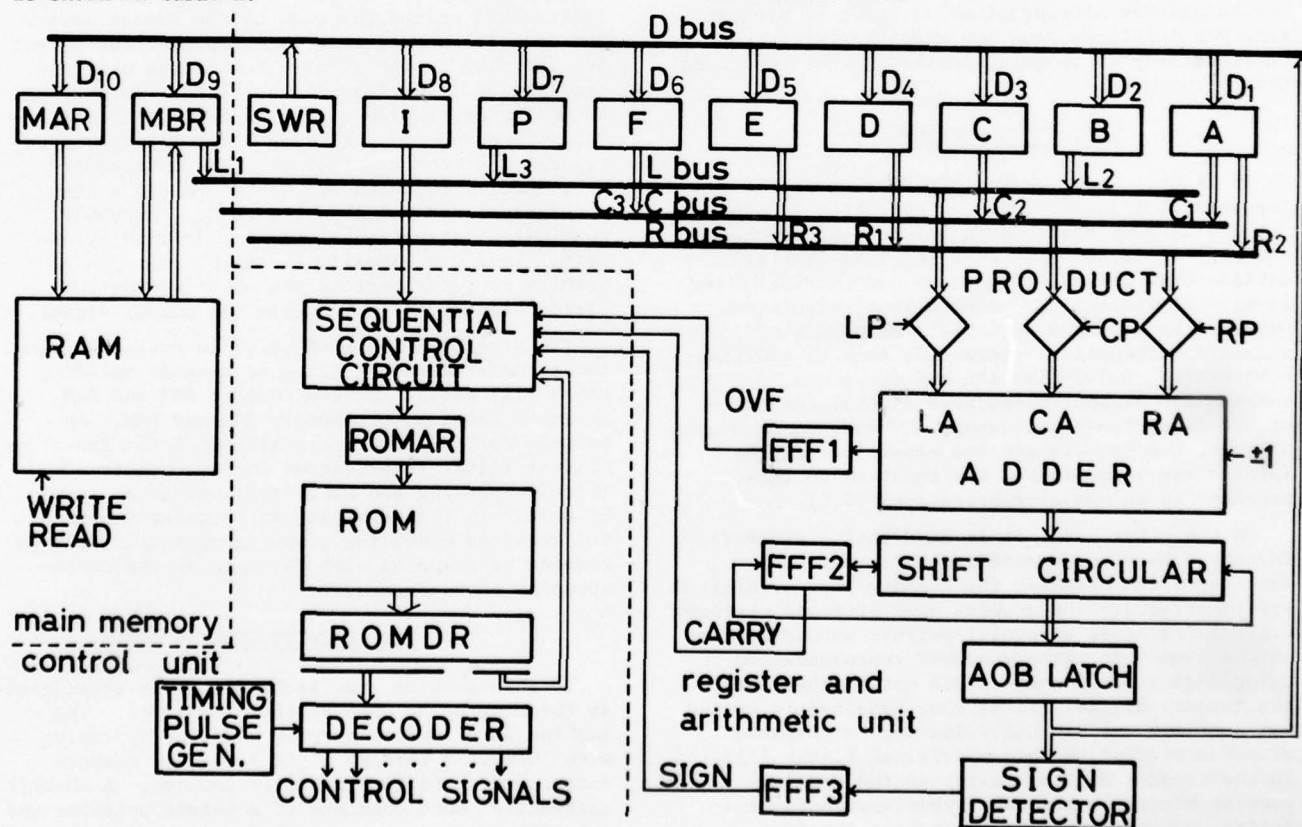


Fig. 1

Register and Arithmetic Unit

This unit has several ternary registers which consist of six 10-trit general-purpose registers (A-F), one 6-trit register for program counter (P), and one 4-trit instruction register (I). Signal fed from the register to the bus line L, C, or R is multiplied by +1, 0 or -1 every trit at the product circuit where the product operation is defined by Table 2, and then is entered to an adder with three inputs. Addition and subtraction can easily be done using the product circuit.

In carrying out digital filtering, many-input addition is frequently required. The three-input parallel adder which consists of ten ternary full adders with four inputs including a carry input as shown in Table 3 is very useful for digital filtering. Unlike the case of the binary adder [14], it should be noted that the three-input addition can be done by only one stage of the ternary full adders. In the adder, a control signal +1, 0 or -1 is entered to the ternary full adder which has the smallest weight as a carry input. As a result, the up-down counting can easily be done by the control signal.

Table 2

X \ Y	1	0	-1
1	1	0	-1
0	0	0	0
-1	-1	0	1

Table 3

$\sum_{i=1}^3 X_i + C$	4	3	2	1	0	-1	-2	-3	-4
Carry	1	1	1	0	0	0	-1	-1	-1
Sum	1	0	-1	1	0	-1	1	0	-1

The output of the adder is transferred to a latch circuit for the adder output buffer (AOB) through a shift circular circuit which operates as left and right shifting. In addition, a sign detector, and flags for overflow, carry and sign (FFF1, FFF2, and FFF3) are included. By testing the values stored in these 1-trit ternary registers, 3-valued branching can be executed by one microprogram step. A switch register (SWR) is used for entering program and data into the ternary computer.

Main Memory

The main memory is composed of $3^6 = 729$ words, each of them having 10 trits. One word can be used to store one ternary number or one instruction of the programs, coded in ternary form. Some words are allocated to the I/O registers. The memory address register (MAR) and the memory buffer register (MBR) are 6-trit and 10-trit widths, respectively. The content of the memory location specified by the MAR is read-out to the MBR. On the other hand, the content of the MBR is stored into the memory location specified by the MAR.

Control Unit

This computer which may be classified as a horizontal machine employs the microprogrammed control. The microinstruction width is 23 trits. The ternary ROM which serves as the store for the microprogram is composed of $3^5 = 243$ words, each of them having 23 trits. The content of the memory location specified by the read-only memory address register (ROMAR) is read-out into the read-only memory data register (ROMDR) every machine cycle. The ROMAR and ROMDR are 5-trit and 23-trit widths, respectively.

IV. MICROPROGRAMMING

Microinstruction Format

The microinstruction format is depicted in Fig. 2. The microinstruction as defined here requires 23 trits. The microinstruction fields have two groups: the static and sequential control fields.

Static Control Fields												Sequential Control Fields				
W R	S L	S C	S R	L P	C R	P P	± 1	C	C	S H	C	D	B T	F B	N	A

Fig. 2

The static control field can be divided into the following mutually exclusive fields: W/R, SL, SC, SR, LP, CP, RP, ± 1 , CC, SH, C, D. By using ternary logic, many fields are defined by only one trit, so that a control unit can be simplified.

The sequential control fields specify the address of the next microinstruction. If the conditional branch test is required, then the operation for modifying the sequencing scheme based on the results of the branch test specified is carried out. Three fields are used here: BT, FB, NA.

Static Control Field

(1) Write/Read field, W/R (1 trit)

This field controls the main memory:

- 1 Write
- 0 No operation
- 1 Read.

(2) Data selection for the L, C, and R bus lines, SL, SC, and SR (1 trit for each)

These fields select the registers for the L, C, and R bus lines.

(3) Product fields, LP, CP, and RP (1 trit for each)

These fields control the product circuit and transfer the data source to the inputs of the adder:

LP	CP	RP
1 L bus+LA	1 C bus+CA	1 R bus+RA
0 "0" +LA	0 "0" +CA	0 "0" +RA
-1 L bus+LA	-1 C bus+CA	-1 R bus+RA

- (4) Lowest carry-input field, ± 1 (1 trit)

The up-down counting can easily be performed by this field :

- 1 Add with "+1" carry-in to the lowest full adder
- 0 Add with " 0" carry-in to the lowest full adder
- 1 Add with "-1" carry-in to the lowest full adder.

- (5) Carry control field, CC (2 trits)

This field controls the carry register FFF2.

- (6) Shift circular control field, SH (1 trit)

A 1-trit field is sufficient to encode the three possibility :

- 1 Shift right one trit
- 0 No shift
- 1 Shift left one trit.

- (7) Overflow flag field, C (1 trit)

This field controls the OVF register FFF1.

- (8) Data bus destination, D (3 trits)

Information on the data bus is routed to the destination specified by this field.

Sequential Control Fields

- (1) Branch test field, BT (2 trits)

Conditions from which to determine the lowest address trit for the next microinstruction address are specified by this field. Using this field, 3-valued branching can easily be executed. One must designate the lowest trit of the next address to be "0" :

- | | | |
|-------|--------------|--|
| 0 0 | No test | |
| 1 1 | OVF test | Replace the address trit with the value of FFF1. |
| 1 0 | Carry test | Replace the address trit with the value of FFF2. |
| 1 -1 | Sign test | Replace the address trit with the value of FFF3. |
| -1 1 | OVF 1 test | Set the address trit to "1" if the value of FFF1 is "1", and otherwise set "0". |
| -1 0 | Carry 1 test | Set the address trit by the value of FFF2 as mentioned above. |
| -1 -1 | Sign 1 test | Set the address trit by the value of FFF3 as mentioned above. |
| 0 -1 | Sign 0 test | Set the address trit to "0" if the value of FFF3 is "0", and otherwise set "-1". |

- (2) Functional branch field, FB (1 trit)

Machine-language instructions can be

connected with the microprogram by this field.

- (3) Next address field, NA (5 trits)

These trits directly supply the address for the next microinstruction to be executed.

Microprogramming Timing

Fig. 3 shows the time chart in execution of a microinstruction. One microinstruction is executed in one machine cycle. The content of the ROM specified by the ROMAR is read-out by a pulse C_1 . The microinstruction decoded controls the corresponding gates. Let the decoding time be t_1 , the delay time of the product circuit be t_2 , the time required for the adder be t_3 , and the delay time of the shift circular circuit be t_4 . If the propagation delay time of the gates used here is assumed to be τ sec, then $t_1 = 8\tau$, $t_2 = 2\tau$, $t_3 = 21\tau$, and $t_4 = 2\tau$. After a elapsed time ($t_1 + t_2 + t_3 + t_4 + \Delta t$), a set pulse C_2 must be sent to the AOB latch circuit.

The elapsed time determined by t_3 is substantially decreased as compared with the binary case because of the reduction of the digits and the stage of gates needed. For example, if the propagation delay time of binary AND, OR and NOT gates is assumed to be τ sec as well as the case of the ternary gates, then the time required for the corresponding 16-bit 2-input adder will be 35τ . A pulse C_3 set the registers, and then a pulse C_4 the ROMAR.

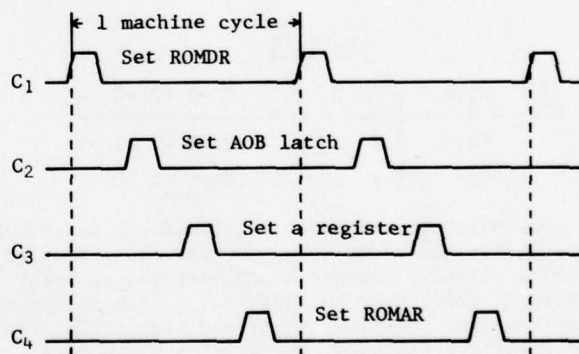


Fig. 3

Microprogramming Example

The machine instruction consists of a 4-trit operator and a 6-trit operand. Fig. 4 represents the flow chart for stage I (fetch cycle). The microprogramming of stage II (execution cycle) is given by the machine instruction used. As an example of microprogramming, consider the case of multiplication which determines the operating time of digital filters. The flow chart is shown in

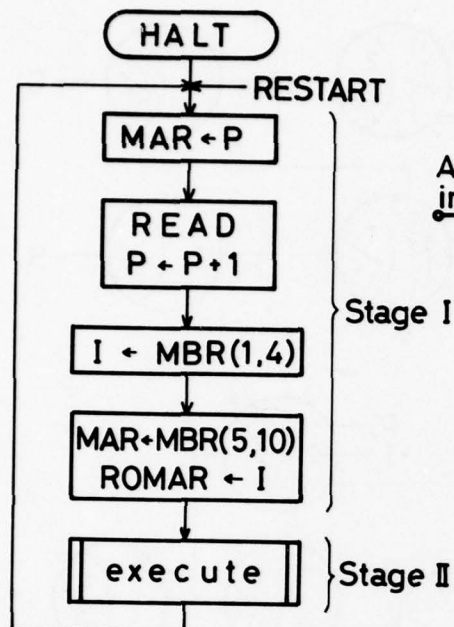


Fig. 4

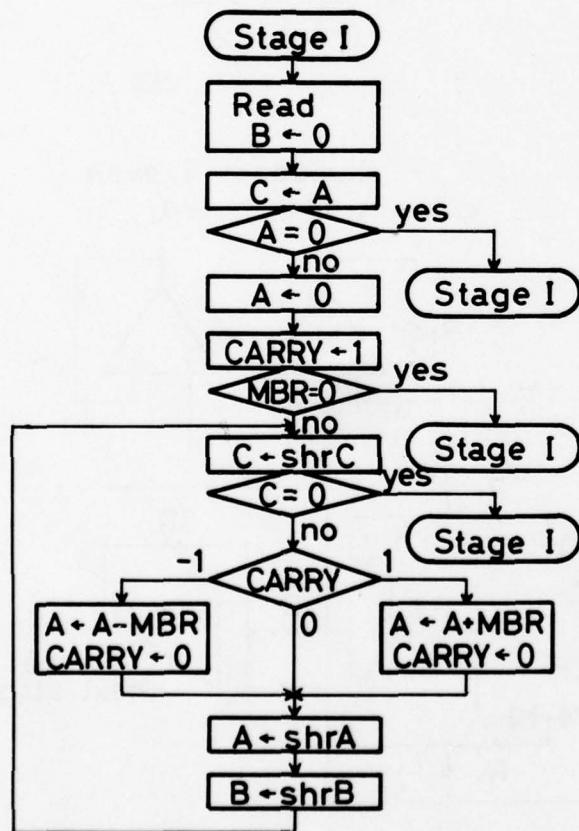


Fig. 5

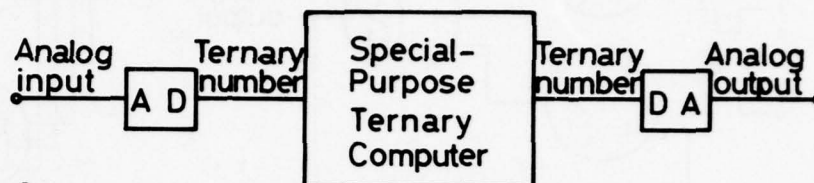


Fig. 6

Fig. 5. From this figure it follows that micro-programming for the ternary multiplication can simple be done without a sign conversion and complex overflow detection. The multiplier (MBR) and multiplicand (A register) are 10 trits, and the product obtained will be 20 trits, (A-B registers). In rounding off the result into 10 trits, no special round off technique is required. The microprogramming steps necessary for the ternary multiplication are 13 steps. On the other hand, 27 microprogram steps are required for the case of the corresponding binary computer which must use more complex algorithms.

V. PRACTICAL IMPLEMENTATION

Fig. 6 shows a diagram of the real-time digital filter under consideration. In the digital filter, the signal to be processed is an analog signal, while the processing is performed digitally. Therefore, the practical implementation needs the A/D and D/A converters as well as the special-purpose ternary computer.

First, consider the ternary computer. The practical implementation is realized using the ternary threshold gates as a basic building block [11] except for the ternary ROM and RAM realized based on the binary ROM and RAM. The basic circuits utilized for the implementation are the DFFF [15], the four-input full adder including a carry input [11], the product circuit, and the T-gate as shown in Figs. 7, 8, 9 and 10, respectively.

The ternary A/D converter used here has been realized on the basis of the commercially available IC operational amplifier [17]. Fig. 11 shows the flow chart representing the algorithms for the A/D converter where X is the analog input and a_i is the i th-trit. Fig. 12 represents the circuit realization based on the flow chart of Fig. 11. Similarly, the D/A converter is realized based on ladder networks [18].

The ternary ROM and RAM are implemented based on the commercially available binary ROM and RAM, but it is omitted here.

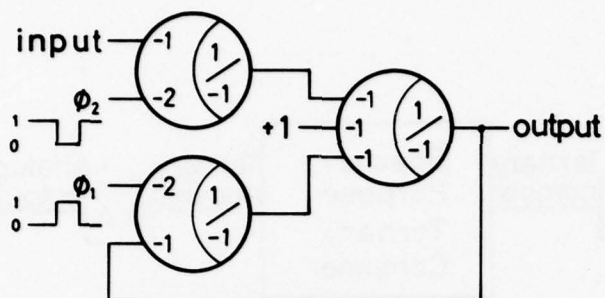


Fig. 7

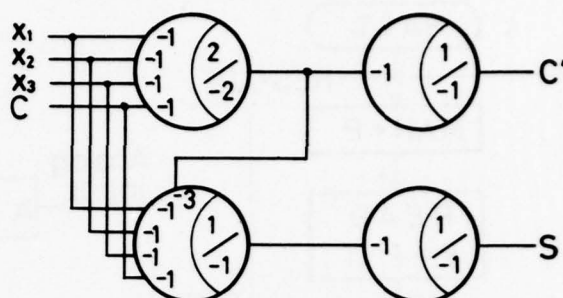


Fig. 8

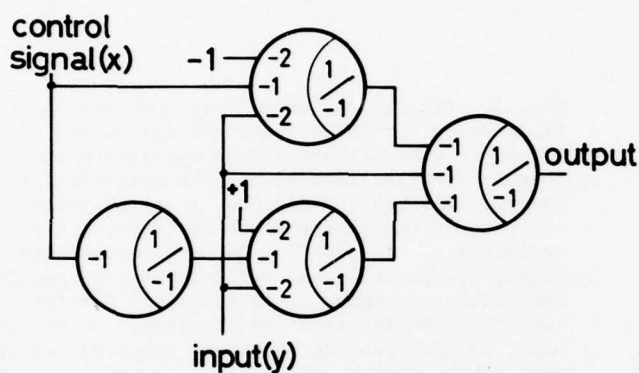


Fig. 9

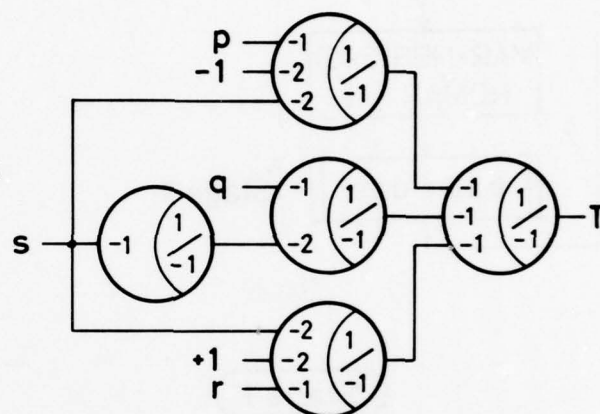


Fig. 10

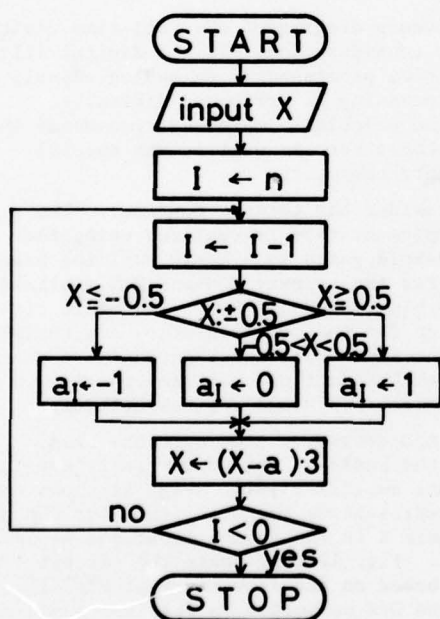


Fig. 11

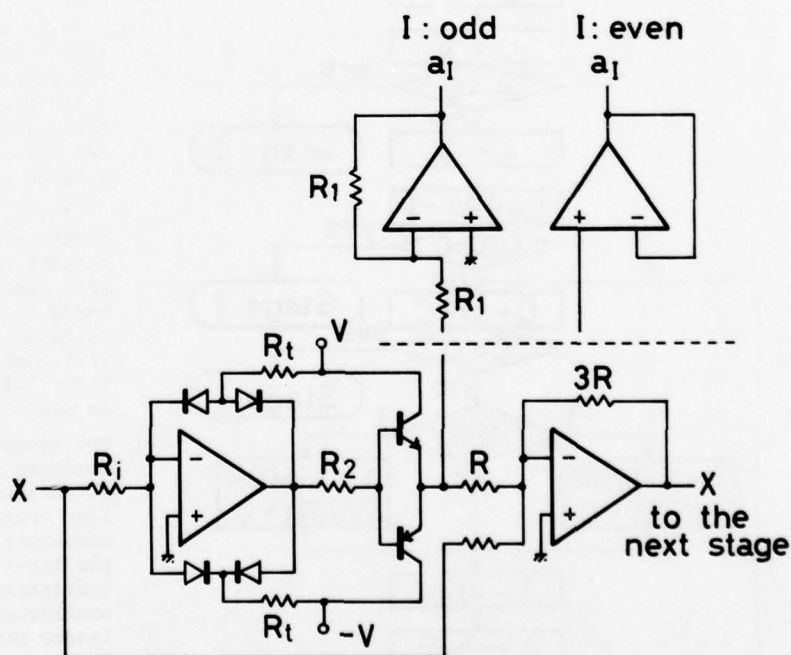


Fig. 12

VI. APPLICATION TO DIGITAL FILTERING

As a simple example, consider an implementation of the canonic form of the 2nd-order digital filter of Fig. 13. This structure forms the basic building block for the development of further and more complicated canonic structures. Any nth-order digital filter can be represented as either a cascade or a parallel (or combination of both) of this structure.

Digital filtering considered here must carry out two 3-input additions, five multiplications, and two delay operations. The 3-input addition can be carried out in one machine cycle as well as the 2-input addition. The detection of overflow and underflow is more elegant and simple because the ternary addition can be done in one step, and needs no complement representation. The flow chart for the 3-input addition is illustrated in Fig. 14. If overflow or underflow occurs, the result is set to the maximum or minimum value, so that overflow oscillation can be avoided [13]. In the binary case, it is practically impossible to detect strictly overflow in the 3-input addition because of awful complexity.

The microprogramming steps for the multiplication mentioned earlier are reduced to about a half of the ones for the binary multiplication. The execution steps which determine the operating time of the digital filter are also decreased. In the ternary and binary multiplication, the machine cycles needed are $5t + 8$ and $5b + 21$, respectively where t is the number of trits and b is the number of bits. The relation between t and b becomes $t = 0.63b$. Hence, the execution steps in the ternary case are reduced to $3.15b + 8$. For example, the 10×10 -trit multiplication can be carried out by only 58 execution steps, while the corresponding 16×16 -bit multiplication needs 101 steps. Fig. 15 represents the execution steps required for the binary and ternary multiplication. This figure shows that the required execution steps can substantially reduced as compared with the case of the corresponding binary multiplication.

VII. CONCLUSION

This paper has described the design and implementation of a special-purpose microprogram-controlled ternary computer for digital filtering. It has been demonstrated that many advantages of the ternary number system can effectively be employed for the computer in hardware and software. In particular, it has been shown that the ternary number system simplifies the arithmetic operations required for digital filters, so that the hardware structure and programming for the ternary computer can easily be carried out.

ACKNOWLEDGMENT

The authors wish to thank Prof. T. Anayama for many helpful discussions.

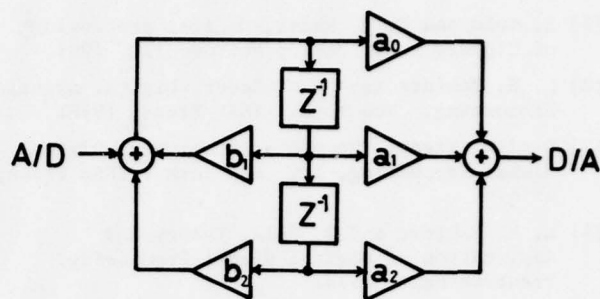


Fig. 13

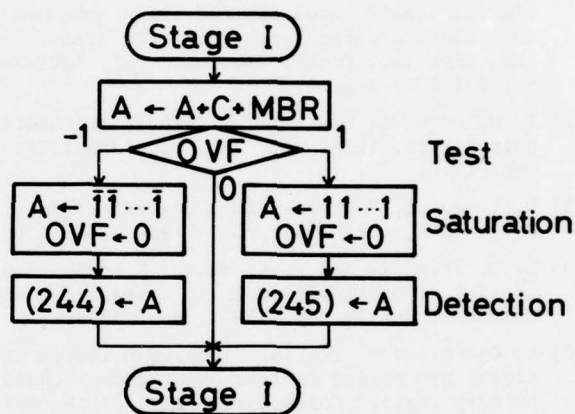


Fig. 14

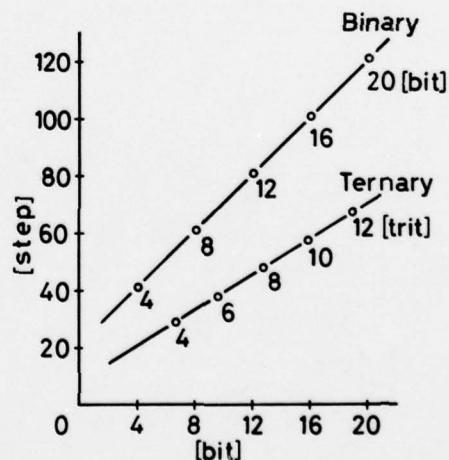


Fig. 15

REFERENCES

- [1] B. Gold and C. M. Rader, Digital processing of Signals. New York : McGraw-Hill, 1969.
- [2] L. R. Rabiner and C. M. Rader, Digital Signal Processing. New York : IEEE Press, 1972.
- [3] Digital Signal Processing Committee, Digital Signal Processing, II. New York : IEEE Press, 1976.
- [4] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing. Prentice-Hall, 1975.
- [5] N. Ahmed and J. P. Jayapalan, " On digital filter implementation via microprocessors," IEEE Trans. Ind. Electron. Contr. Instrum., vol. IECI-23, pp. 249-253, Aug. 1976.
- [6] T. Higuchi, T. Saito and A. Kanomata, " A microprocessor-based digital filter programmed in a block diagram language," IEEE Trans. Ind. Electron. Contr. Instrum., vol. IECI-24, pp. 231-234, Aug. 1977.
- [7] I. Halpern and M. Yoeli, " Ternary arithmetic unit," Proc. IEEE, vol. 115, pp. 1385-1388, Oct. 1968.
- [8] D. I. Porat, " Three-valued digital systems," Proc. IEEE, vol. 116, pp. 947-954, June 1969.
- [9] Z. G. Vranesic and K. C. Smith, " Engineering aspects of multi-valued logic systems," IEEE Computer, vol. 7, pp. 34-41, Sep. 1974.
- [10] L. Ojala and P. Yrjölä, " Design of incremental signal processing devices using binary coded ternary logic," Conference Record of the 1973 International Symposium on Multiple-valued Logic, Toronto, May 24-25, 1973.
- [11] T. Higuchi and K. Kobayashi, " Hardware implementation of digital filter based on ternary threshold gates," Proceedings of the 7th International Symposium on Multiple-valued Logic, Charlotte, May 24-27, 1977.
- [12] D. C. Rine, Computer Science and Multiple-valued Logic, theory and applications. North-Holland Publishing Company, 1977.
- [13] P. M. Ebert, J. E. Mazo and M. G. Taylor, " Overflow oscillations in digital filter," Bell Syst. Tech. J., vol. 38, pp. 2999-3020, Nov. 1969.
- [14] G. L. Kratz, W. W. Sproul and E. T. Walendziewicz, " A microprogrammed approach to signal processing," IEEE Trans. Computers, vol. C-23, pp. 808-817, Aug. 1974.
- [15] T. Higuchi and M. Kameyama, " Static-hazard-free T-gate for ternary memory element and its application to ternary counters," IEEE Trans. Computers, vol. C-26, pp. 1212-1221, Dec. 1977.
- [16] T. Higuchi and M. Kameyama, " Ternary logic system based on T-gate," Proceedings of the 5th International Symposium on Multiple-valued Logic, Bloomington, May 13-16, 1975.
- [17] K. Kobayashi and T. Higuchi, " A/D converter for ternary digital filter," 1977 National Convention Record of the Institute of Electronics and Communication Engineers of Japan, 31, Aug. 1977 (in Japanese).
- [18] L. Ojala and P. Yrjölä, " Design of A/D converters for binary coded ternary systems," Conference Record of the 1972 Symposium on the Theory and Applications of Multiple-valued Logic Design, New York, May 25-26, 1972.

DESIGN AND IMPLEMENTATION OF A NON-BINARY CODE FOR BYTE-ORGANIZED MEMORY WITH BINARY AND QUATERNARY LOGICS*

Tich T. Dao

Signetics Corporation
Sunnyvale, California 94086

ABSTRACT

Byte-organized memory requires an error control scheme which can handle errors involving one or several entire bytes. A special parallel non-binary single error correction and double error detection SEC-DED block code is constructed by an iterative technique. This code is optimum in the sense that it lends itself to a simple and high speed hardware implementation either in binary or in quaternary logic. A double extension field $GF(2^{2m})$ of the subfield $GF(2^m)$ is then introduced. As a design example a (80,64) SED-DED code in $GF(2^4)$ is constructed and implemented.

INTRODUCTION

New semiconductor memory systems are being developed which offer significant size, speed and weight advantages over conventional magnetic memory systems. With the availability of high-speed and inexpensive logic, some type of error detection and correction is incorporated into the system in order to enhance its reliability. It is usually a modified version of Hamming code designed for optimum hardware implementation.

An optimum code must satisfy several important considerations:

- 1) Speed: error correction should be completed in a very small fraction of the available CPU cycle time.
- 2) System compatibility: information processing and transfer in and out of a CPU are performed in a parallel, not (as in communication channels) in a serial mode. The error correction code must be compatible with the memory system organization.
- 3) Encoder-Decoder reliability: a requirement

*A large part of this work has been done while the author was with Ampex Corp.

for parallel operation and high speed could result in a complex encoder-decoder design with subsequent reliability problems which must be resolved in the design stage.

- 4) Cost: incorporation of error correction into a system will necessarily increase the overall cost in terms of additional data redundancy, computing time and hardware.

Organization of the memory, failure mode of the storage cell and of the memory subsystem will determine the statistics of the errors in the system and consequently the choice of the code. Overall system design is greatly improved by an understanding of these interrelationships.

It has been proved both in the IBM 370 systems^{1,2,3,4} and the Nova/Eclipse that a modified Hamming SEC-DED code can be applied efficiently to a memory system organized in such a way that errors occurring during readout are random errors, but not burst errors. To handle random errors only, a distinct memory must be physically assigned to each bit of the word, each bit having its own address decoder, drivers and sensing circuitry. This memory organization is referred to as the Bit Per Operating Memory (Bit/BOM) organization. The obvious advantage of such a structure is that a single failure in either the storage cell, the decoder, or sense amplifiers can now be constrained to affect, at most, a single bit of the data word.

A second organization called Byte Per Operating Memory (Byte/BOM) (Fig. 1) lends itself to a different mode of errors. A failure on a single wafer, involving either the storage cell or the sensing amplifier, would affect a single bit in the byte but a failure in the associated overhead circuitry would alter all the bits in the byte. A memory organized in Byte/BOM would have, therefore, random bit errors, and also random byte errors, the latter being a special case of the conventional burst error. To combat this failure mode, one must use a code that could correct random errors as well as burst errors, and which

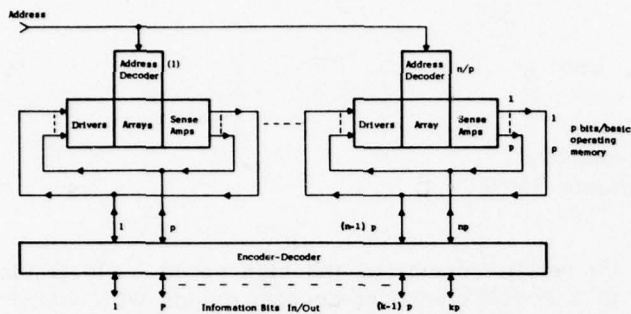


Fig. 1 Byte Organized Memory System Byte/BOM

also lends itself to high-speed parallel implementation.

The type of code we are going to deal with is the non-binary code in $GF(2^p)$, with its subclass the generalized Hamming code. More specifically, we will present both binary and quaternary implementations of a non-binary SEC-DED code appropriated for a byte-organized memory.

NON-BINARY HAMMING CODE

As in the case of the binary code, a linear non-binary block code can be uniquely defined by a parity matrix.² The elements of the matrix are elements of the $GF(q)$ in question.

Consider a SEC-DEC (n, k) code in $GF(2^p)$. Since it has N bytes' length, with K bytes of information, and M bytes of parity check with $N = M + K$, then the H matrix has $N \times M$ elements in $GF(2^p)$, $(N = \frac{n}{p}, K = \frac{k}{p}, \text{ and } M = \frac{m}{p})$.

Formally, the H matrix can be represented by N columns, C_1 to C_n of K elements each.

$$H = C_1, C_2, \dots, C_j, \dots, C_n \quad \text{with } C_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{kj} \end{bmatrix}$$

The theorem regarding the H matrix of a binary SEC-DED still holds true in the case of a non-binary. We repeat here:

THEOREM 1: A code defined by the parity matrix H will correct a single byte error and detect a double byte error if and only if every

combination of 3 or fewer columns of H is linearly independent.

A single error of value e occurring at the byte position j produces a column syndrome of value

$$S_j = eC_j = \begin{bmatrix} ea_{1j} \\ \vdots \\ ea_{kj} \end{bmatrix} \quad \text{for all } e \text{ in } GF(2^p)$$

e being an arbitrary element of $GF(2^p)$. Therefore, all possible multiples of the column C_j represent the syndrome of the error byte at position j : set S_j . If the multiples of C_j are chosen such that all the syndrome sets S_j ($j=1 \dots N$) are disjoint, then there exists a one to one correspondence between the error position and the syndrome set (isomorphism). Conversely, the error byte position is located as soon as the computed syndrome is identified with its proper set:

since

$$S_i \cap S_j = \emptyset \text{ if } \beta C_i \neq \lambda C_j \text{ and } \beta, \lambda \text{ in } GF(2^p)$$

Explicitly, this means that any two syndrome sets S_i, S_j (for all $i \neq j$) are disjoint if and only if the two corresponding column vectors C_i, C_j are non-collinear (mod 2 addition $\neq 0$). The column vectors of H must be all distinct from each other.

Furthermore, the code is double-error detected if the sum of any two syndrome sets corresponding to two single-error bytes $(S_i \oplus S_j)$, ($i \neq j$) is not equal to any single-error syndrome set S_k .

$$S_i \oplus S_j \neq S_k \text{ for all } S_i, S_j, S_k \text{ in } S \text{ with } S_i \neq S_j$$

Since $S_i = e_1 C_i$, $S_j = e_2 C_j$, and $S_k = e_3 C_k$ for some e_1, e_2, e_3 in $GF(2^p)$, then explicitly we can write

$$e_1 C_i \oplus e_2 C_j \oplus e_3 C_k \neq e_1 * C_i \oplus e_2 * C_j \neq C_k$$

for some e_1^*, e_2^* in $GF(2^p)$.

This means that every combination of 3 columns of H (i.e., C_i, C_j, C_k) is linearly independent. If not, then the matrix formed by the 3 columns must be identically zero; in other words, any one vector can be obtained by a linear combination of the two other columns.

If we design the H matrix in such a way that the first non-zero element of the syndrome column S computed from the received word gives

directly the value of the error e , then the error byte location is reduced to the search of a column C_1 of H which verifies the identity $eC_1 = S$. This scheme is proposed by Ulrich.⁵

LEMMA 2.1: No computation of the error value is required if the first non-zero element of every column of H is a unit element α^0 .

Of course, the same holds true when the column is scanned upwards. The proof is immediate. Since $e\alpha^0 = e$, the first non-zero element of the syndrome is therefore equal to e .

According to W. W. Peterson, "Every linear code is equivalent to a systematic code." Theorem 2.²

The H matrix can be put in the form $H = [P, I_m]$ where I_m is a unit matrix ($m \times m$) and P is the proper parity matrix ($k \times m$). If the weight of a column is defined as the number of its non-zero elements, then the following lemma applies:

LEMMA 2.2: H is a parity matrix of a SEC-DED code only if every column of P has a weight of at least three.

Proof: If not, then any column of weight 2 always can be obtained by a linear combination of two columns of I_m . The three columns in question then form a linear dependent group. This is not the case.

The optimum H matrix must satisfy LEMMA 2.1 and LEMMA 2.2. The following conditions apply to every column of P :

- 1) Each column has a minimum number of non-zero elements;
- 2) these elements are mostly unit elements α^0 and 0;
- 3) all other elements ($\neq 0$ and $\neq \alpha^0$) belong to a minimum subset of the $GF(2^p)$ in question. This restriction
 - 1) provides for minimum arithmetic operation required;
 - 2) reduces particularly the number of multiplication operations;
 - 3) reduces the number of different types of multiplication operators needed.

Furthermore, from the hardware point of view, the optimum matrix is that which minimizes the minimum number of multiplication and addition operations necessary to generate either the

parity check bits or the syndrome. With the non-binary code, correction requires not only location of the error, but also computation of the error value before correction. This additional step, which does not exist in binary codes, must be taken into consideration.

Application: Let us apply the above rules to the design of a parity matrix of an (80, 64) SEC-DED code in $GF(2^4)$, where elements are 0, 1, 2, $2^2 \dots 2^{14}$, and whose sum and product operators are defined by tables 1 and 2. The matrix has 4×20 elements; each column has at least 0 and α^0 (LEMMA 2.1); each column (not included in the section I_4) must have 3 non-zero elements (LEMMA 2.2).

- 1) As usual, we start with the unit matrix I_4 .
- 2) To it we add 4 columns, which are all possible permutations of the set $(\alpha^0, \alpha^0, \alpha^0, 0)$. It can be verified that the 8 column matrix so obtained constitutes a valid and optimum submatrix.

3) A group of 8 columns made of all possible combinations of the set $(0, \alpha^0, \alpha, \alpha^2)$ with α^0 restricted to be at the 1st or 2nd position, can be added to the above submatrix. The restriction on α^0 is due to the Lemma 2.1. The reason for the use of two new elements α, α^2 rather than one new element only is necessitated by the linear independency condition. Assume we have a column which is, for example:

$$\begin{bmatrix} \alpha^0 \\ 0 \\ \alpha^0 \\ \alpha \end{bmatrix}$$

Table 1
Operation in $GF(2^2)$

	Addition				Multiplication				Binary Representation	
	0	α^0	α^1	α^2	0	α^0	α^1	α^2	0	0
0	0	α^0	α^1	α^2	0	0	0	0	0	0
α^0	α^0	0	α^2	α^1	α^0	0	α^0	α^1	α^0	1
α^1	α^1	α^2	0	α^0	α^1	0	α^1	α^2	α^1	0
α^2	α^2	α^1	α^0	0	α^2	0	α^2	α^0	α^2	1

It can be verified that:

$$\begin{bmatrix} \alpha^0 \\ 0 \\ \alpha^0 \\ \alpha \end{bmatrix} + \begin{bmatrix} \alpha^0 \\ 0 \\ \alpha^0 \\ \alpha^0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \alpha + \alpha^0 \end{bmatrix} = \alpha^4 \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ \alpha^0 \end{bmatrix}$$

Therefore the new column could be obtained by a linear combination of two existing columns.

4) More restricted permutations of the set $0, \alpha^0, \alpha^3, \alpha^7$ are added to 16 columns so far obtained to complete the parity matrix as shown in Table 3. The fact that we have to use two new elements follows from the previous observations. A computer search of all possible single error and double-error combinations has proved the validity of the matrix. It is, by construction, also optimal, according to the principle of optimality, enunciated as follows by Bellman⁴ in his dynamic programming procedure. "An optimal set of decision has the property that whatever the first decision is, the remaining decisions must be optimal with respect to the outcome which results from the first decision."

Loosely speaking in the context of the SECDED coding theory, the optimum $(k \times n)$ matrix M_n^k is obtained from an optimum $k \times (n-1)$ matrix $M_{(n-1)}^k$ by adding to the $M_{(n-1)}^k$ a column vector C_1 such that:

$$M_n^k = \text{minimum for all } i \left[M_{(n-1)}^k + C_1 \right]$$

with the following constraints:

a) C_1 has at least 3 non-zero elements; its top element is either "zero" or "one;"

b) any group of 3 columns in $M_{(n-1)}^k$ and M_n^k is linearly independent;

and with the criteria of optimality being as follows:

a) The M_n^k has maximum 0 and maximum α^0 .

b) Other elements in M_n^k belong to the smallest subset of the $GF(2^p)$.

Table 2 Arithmetic Operations in $GF(2^4)$

ADDITION																MULTIPLICATION																BINARY REPRESENTATION																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
Arithmetic Operations in GF(2)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	0	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	0	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
1	1	0	3	2	7	6	5	4	F	E	D	C	B	A	9	8	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
2	2	3	0	1	6	7	F	E	4	5	D	C	B	A	9	8	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
3	3	2	1	0	F	E	4	5	D	C	B	A	9	8	0	0	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
4	4	5	F	E	0	1	2	3	6	7	8	9	A	B	C	D	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
5	5	6	E	D	7	6	5	4	0	1	2	3	4	5	6	7	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
6	6	7	F	E	4	5	D	C	B	A	9	8	0	0	1	2	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
7	7	8	E	D	6	5	4	0	1	2	3	4	5	6	7	8	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
9	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
A	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
B	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
C	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
D	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
E	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
F	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	0	1	2	3	4	5	6	7	8	9	A	B	C	D																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

ENCODING AND DECODING OPERATIONS OF AN SEC-DED CODE IN GF(2^P)

Encoding

To show how the encoding and decoding of an SEC-DED code in GF(2^P) are performed, let us take a concrete example of the (80,64) code in GF(2⁴) whose parity check matrix H is given in Table 3. There are 20 bytes (4 bits/byte) in the code. The first 16 bytes are information bytes. The last four are parity bytes. Symbolically the code is represented by:

$$B_1, B_2, B_3, B_4, B_5, B_6, \dots, B_{20}$$

During encoding, the information bytes are given B_1, \dots, B_{16} , and the parity bytes B_{17}, \dots, B_{20} are computed from the corresponding H matrix. Explicitly, parity bytes are defined by the following set of simultaneous linear equations of elements in GF(2⁴):

$$A \begin{cases} B_{17} = B_1 + B_2 + B_3 + B_5 + B_6 + B_7 + B_9 + B_{10} + B_{11} + B_{13} + B_{14} + B_{15} \\ B_{18} = (B_1 + B_2 + B_4 + B_8 + B_{12} + B_{16}) + (B_5 + B_7) \alpha + (B_9 + B_{11}) \alpha^2 + (B_{13} + B_{14}) \alpha^3 \\ B_{19} = (B_1 + B_3 + B_4) + (B_6 + B_8 + B_9) \alpha + (B_5 + B_{10} + B_{12}) \alpha^2 + (B_{15} + B_{16}) \alpha^3 + B_{13} \alpha^7 \\ B_{20} = (B_2 + B_3 + B_4) + (B_{10} + B_4 + B_{12}) \alpha + (B_6 + B_7 + B_8) \alpha^2 + (B_{14} + B_{15} + B_{16}) \alpha^7 \end{cases}$$

1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	0	0	0
1	1	0	1	α	0	α	1	α^2	0	α^2	1	α^3	α^3	0	1	0	1	0	0
1	0	1	1	α^2	α	0	α	α	α^2	0	α^2	α^7	0	α^3	α^3	0	0	1	0
0	1	1	1	0	α^2	α^2	α^2	0	α	α	α	0	α^7	α^7	α^7	0	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Decoding

The syndrome (S_1, S_2, S_3, S_4) is obtained by adding bitwise mod 2 the computed parity bytes to the received parity bytes. Then each byte S_i is tested for its zero value. There are several cases:

- 1) If all 4 bytes are zero, then no error is detected.
- 2) If an even number of bytes are zero, then a double byte error is detected.
- 3) If 3 bytes are zero, then the error occurs at a parity byte. The position of the non-zero element in the syndrome uniquely locates the parity byte in error, and the value of the error e is, in fact, the non-zero element of the syndrome.
- 4) If there is only one zero byte, then further computation is required to determine the error value and the location of the byte in error:

Inspection of the parity matrix in Table 3 suggests the following rules:

Case 1: The first byte of the syndrome is different from zero; then by construction it is also the error value, and error occurs at byte: 1,2,3,5,6,7,9,10,11,13,14,15.

Case 2: The first byte is zero; then the value of the second byte is the error e , and the error location l_3 is at byte 4,8,12,16.

In either case, when the value of e has been extracted, then e is multiplied by the 5 constants $\alpha^0, \alpha^2, \alpha^3, \alpha^7$ to generate the set $Z = e, e\alpha, e\alpha^2, e\alpha^3, e\alpha^7$.

Assume, for the sake of simplicity of computation, and without loss of generality, that the information sequence is given by:

α^0	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}
1000	0100	0010	0001	1100	0110	0011	1101	1010	0101	1110	0111
B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}

α^{12}	α^{13}	α^{14}	α^{15}
1111	1011	1001	0000
B_{13}	B_{14}	B_{15}	B_{16}

Table 3

Optimum Parity Matrix for a (80,64) SEC-DED Code in GF(4)

By referring to Table 2, we see that:

$$B_1, \dots, B_{16} = \alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{14}, \alpha^0$$

Replacing the B_i ($i = 1, \dots, 16$) by their corresponding values we have:

$$\begin{cases} B_{17} = \alpha^0 + \alpha^1 + \alpha^2 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^8 + \alpha^9 + \alpha^{10} + \alpha^{12} + \alpha^{13} + \alpha^{14} = \alpha^{13} \\ B_{18} = (\alpha^0 + \alpha^1 + \alpha^3 + \alpha^7 + \alpha^{11} + 0) + (\alpha^4 + \alpha^6) \alpha + (\alpha^8 + \alpha^{10}) \alpha^2 + (\alpha^{12} + \alpha^{13}) \alpha^3 = \alpha^3 \\ B_{19} = (\alpha^0 + \alpha^2 + \alpha^3) + (\alpha^5 + \alpha^7 + \alpha^8) \alpha + (\alpha^4 + \alpha^9 + \alpha^{11}) \alpha^2 + (\alpha^{14} + 0) \alpha^3 + \alpha^{12} \alpha^7 = \alpha^5 \\ B_{20} = (\alpha^1 + \alpha^2 + \alpha^3) + (\alpha^9 + \alpha^{10} + \alpha^{11}) \alpha + (\alpha^5 + \alpha^6 + \alpha^7) \alpha^2 + (\alpha^{13} + \alpha^{14} + 0) \alpha^7 = \alpha^{10} \end{cases}$$

Each element of Z is compared to each one of the 4 elements of the syndrome $S = S_1, S_2, S_3, S_4$. The outputs of the thirteen simultaneous comparisons are used in a combinatorial network to locate the error byte. For example, assume that the error occurs at byte number 16. Given the syndrome $S = S_1, S_2, S_3, S_4$, and the error value e , the error location is identified to be at the 16 byte, if and only if:

$$\begin{cases} S_1 = 0 \\ S_2 = e \\ S_3 = e\alpha^3 \\ S_4 = e\alpha^7 \end{cases}$$

PARALLEL IMPLEMENTATION WITH GALOIS OPERATORS

A parallel implementation of the encoder-decoder as discussed above is shown in Figs. 2 and 3. The heavy bus line actually represents a multiplex of 4 bit lines (or a byte line). On top of the figure there are 4 parity byte encoders. Each square box with a constant $\alpha, \alpha^2, \alpha^3, \alpha^7$ represents an operator which performs the products of a byte with a constant $\alpha, \alpha^2, \alpha^3, \alpha^7$. The input line marked with a star (*) is the received parity byte used only during decoding to generate the syndrome. All other lines are inputs from corresponding information bytes.

Encoding resulted from the direct implementation of the relation A where the B multiplex are connected to the inputs, and all the B lines received the respective information bytes.

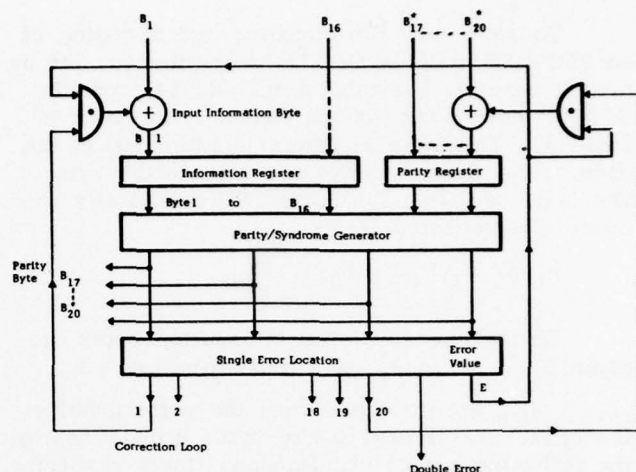


Fig. 2A

Block Diagram of a SEC-DED Encoder/Decoder in $GF(2^4)$

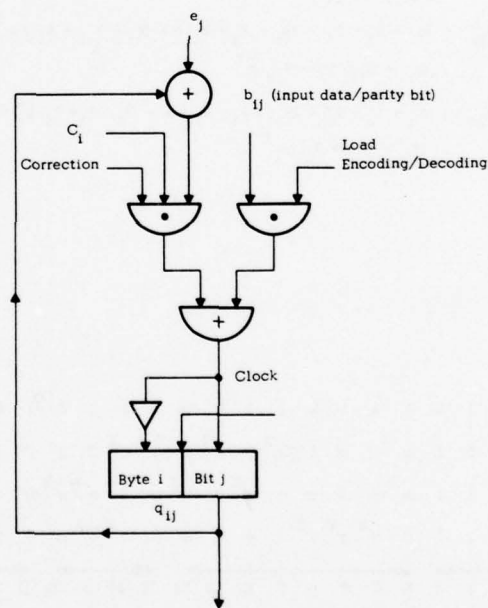


Fig. 2B

Logic Diagram of a Single Bit (B_{ij}) (with its correction loop)

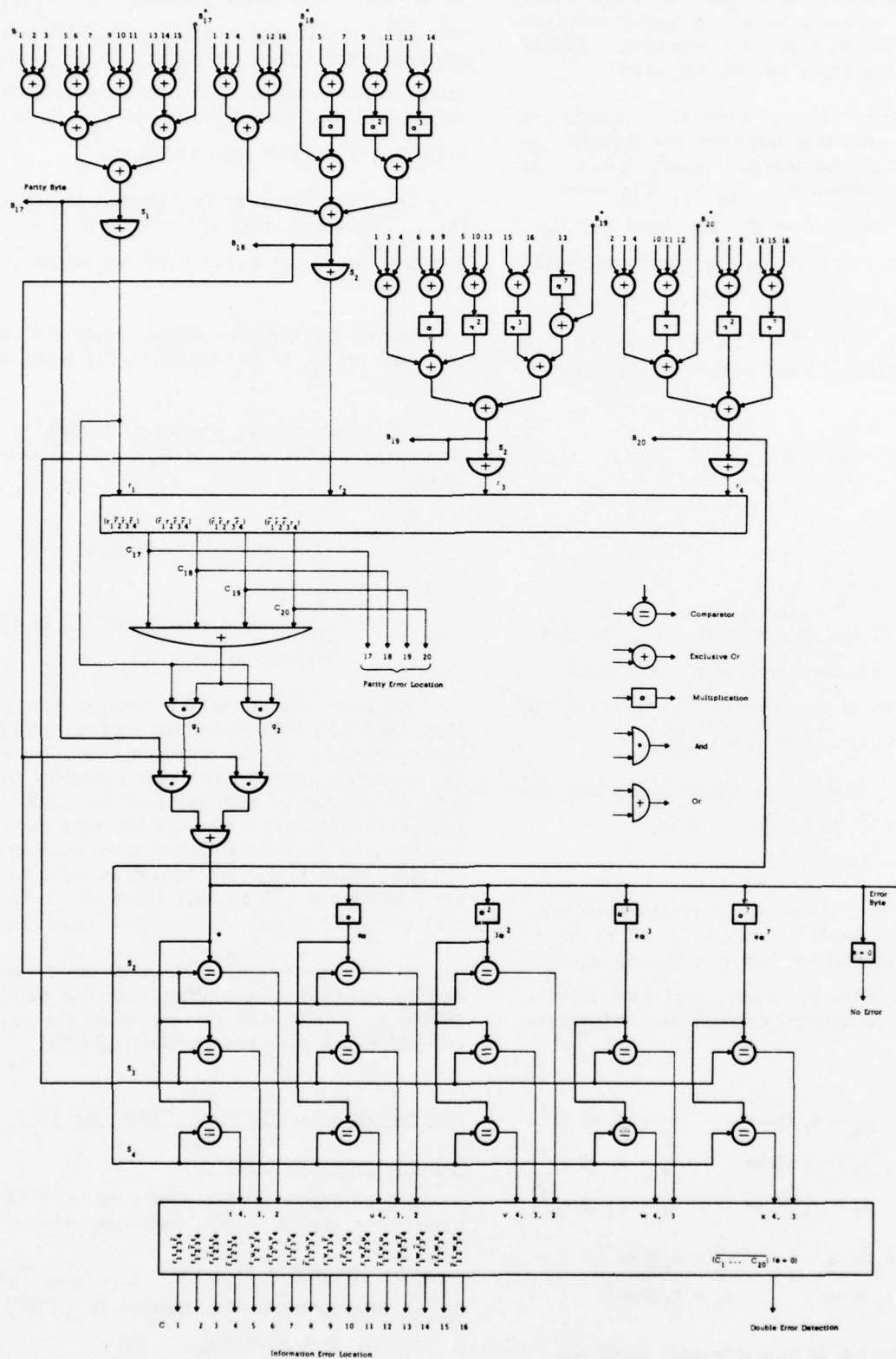


Fig. 3 Logic Diagram of the SEC-DED Encoder/Decoder

The worst delay came from the computation of parity bytes where 3 levels of mod 2 additions plus a multiplication stage are required. Notice that both 2 and 3 input adders are used.

The decoding process goes first through the same stage as encoding but with the B multiplex lines activated by the received parity bytes. At the same points where $B_{17}, B_{18}, B_{19}, B_{20}$ were generated, we obtain now the syndrome $S = S_1, S_2, S_3, S_4$ which is stored in the syndrome register. Outputs r_1, r_2, r_3, r_4 indicate a non-zero value of the S_i element. The $C_{17}, C_{18}, C_{19}, C_{20}$ outputs decide if the error occurs at the parity byte position.

$$\begin{aligned} C_{17} &= r_1 \cdot \bar{r}_2 \cdot \bar{r}_3 \cdot \bar{r}_4 \\ C_{18} &= \bar{r}_1 \cdot r_2 \cdot \bar{r}_3 \cdot \bar{r}_4 \\ C_{19} &= \bar{r}_1 \cdot \bar{r}_2 \cdot r_3 \cdot \bar{r}_4 \\ C_{20} &= \bar{r}_1 \cdot \bar{r}_2 \cdot \bar{r}_3 \cdot r_4 \end{aligned}$$

If neither one of the $C_{17}, C_{18}, C_{19}, C_{20}$ is true, then depending on the value of r_1 , the error occurs at a byte of the group $g_1 = 4, 8, 12, 16$ or $g_2 = 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15$.

If $g_1 = 1$, then S_1 is gated out as the error e . If not, and $g_2 = 1$, then $e = S_2$.

$$e = S_1 g_1 + S_2 g_2$$

A matrix comprising 13 elements each one being made of a 4-bit comparator performs a simultaneous comparison between the S_2, S_3, S_4 or row entries, with the $e, e\alpha, e\alpha^2, e\alpha^3$ column entries. The comparison outputs one defined as:

$$\begin{aligned} t_2 &= S_2 \ominus e & u_2 &= S_2 \ominus e\alpha & v_2 &= S_2 \ominus e\alpha^2 \\ t_3 &= S_3 \ominus e & u_3 &= S_3 \ominus e\alpha & v_3 &= S_3 \ominus e\alpha^2 \\ t_4 &= S_4 \ominus e & u_4 &= S_4 \ominus e\alpha & v_4 &= S_4 \ominus e\alpha^2 \\ w_3 &= S_3 \ominus e\alpha^3 & x_3 &= S_3 \ominus e\alpha^7 \\ w_4 &= S_4 \ominus e\alpha^3 & x_4 &= S_4 \ominus e\alpha^7 \end{aligned}$$

Finally a row of 16 (4 input) gates, each one combining a set of 4 signals from the t, u, v, w, x will identify the error location. Assume now that C_1 is activated, which means that byte 1

is in error. The error sequence $E = e_1, e_2, e_3, e_4$ computed previously must be added bitwise to the corresponding byte $B_i = b_{i1}, b_{i2}, b_{i3}, b_{i4}$ which is in store in the register. The result of the addition constitutes the correct value of B_i . It is then stored at the same byte location.

The input logic of the storage flip-flop register q_{ij} , corresponding to the byte i $i = 1, \dots, 20$ and the bit j $j = 1, 2, 3, 4$ of the register, operates as follows:

During encoding/decoding, after the load control was on, q_{ij} is set equal to the input b_{ij} (Fig. 2b).

When the correct control is turned on, and depending on the value of C_i we have two situations:

- a) $C_i = 0$
 q_{ij} = remains unchanged
- b) $C_i = 1$
 q_{ij} takes a new value \bar{q}_{ij} is the corresponding error bit $e_j = 1$.

We have shown that an optimal binary SED-DED code can be constructed and an efficient implementation can be obtained. We have thus by far implicitly presupposed that hardware to perform operations in any Galois field $GF(2^p)$ exists. This assumption is based on the fact that, according to Dao^{5,6} commercializable components, designed with integrated injection logic technology, are forecoming. They are, however, limited right now to quaternary or four-valued logic family.

It will be proven⁹ that operators in any $GF(2^p)$ can be realized from operators in the subfield $GF(2^2)$ by introducing the notion of the extension field $GF(2^{2m})$ over the subfield $GF(2^m)$.

EXTENSION FIELD $GF(2^m)^2$ OVER $GF(2^m)$

Representation of $GF(2^m)^2$

Let us represent the elements in $GF(2^m)^2$ via binary $2m$ tuples by the following rules.

The $2m$ tuples $(a_0, a_1, \dots, a_{m-1}, b_0, b_1, \dots, b_{m-1})$ correspond to the element in $GF(2^m)^2$

$$\beta = a_0 + Ba_1 \quad (1)$$

where a_0, a_1 are elements in $GF(2^m)$, B is the primitive element in $GF(2^m)^2$, B being the analog

of j the imaginary element in the complex field. From the theory of Galois' field, it is known that if $F(x)$ is a primitive quadratic polynomial over $GF(2^m)$, one can construct the extension field $GF(2^m)^2$ by adding the primitive root B of $F(x)$ to $GF(2^m)$.

$$F(B) = 0$$

$F(x)$ primitive of degree 2 over $GF(2^m)$

and all elements in $GF(2^m)^2$ are represented by (1). On the other hand, all the α_0, α_1 elements of $GF(2^m)$ are represented by

$$\alpha_0 = a_0 + a_1 A + a_2 A^2 + \dots + a_{m-1} A^{m-1}$$

$$\alpha_1 = b_0 + b_1 A + b_2 A^2 + \dots + b_{m-1} A^{m-1}$$

where the a_i, b_i are elements of the binary field $GF(2)$ and A is the primitive element of $GF(2^m)$. In other words, if $F(x)$ is the primitive polynomial of degree m over $GF(2)$, the extension field $GF(2^m)$ is constructed by adding a primitive element A , which is the root of $F(x)$, ($F(A) = 0$). In brief, $GF(2^m)^2$ can be looked at as a double extension of $GF(2)$ via $GF(2^m)$.

Operations in $GF(2^m)^2$

Operations as addition, multiplication, inversion in $GF(2^m)^2$ can be reduced to operations in $GF(2^m)$ via the following rules:

Given β_1, β_2 two non-zero elements in $GF(2^m)^2$:

$$\beta_1 = (a, b) = a + bB$$

$$\beta_2 = (c, d) = c + dB$$

Addition

$$\begin{aligned} \beta_1 + \beta_2 &= (a + bB) + (c + dB) \\ &= (a + c) + (b + d)B \\ &= (a + c, b + d) \end{aligned}$$

Addition is obtained by adding the corresponding components.

Multiplication

Assume that the primitive polynomial $F(x)$ is of the form

$$F(x) = x^2 + x + \delta$$

such that $B^2 + B + \delta = 0$

with $\delta \in GF(2^m)$. Then the product of

$$\begin{aligned} \beta_1 \cdot \beta_2 &= (a + bB)(c + dB) \\ &= (ac + bdB^2 + (ad + bc)B) \\ &= ((ac + bd\delta), (ad + bc + bd)) \end{aligned}$$

Examples: Binary and quaternary implementations

1) $N=4$ or operators in $GF(4)$

The only possible value of δ in $GF(2)$ which satisfies the condition is $\delta = 1$; therefore, we have:

$$\beta_1 \beta_2 = [(a_1 a_2 + b_1 b_2), (a_1 b_2 + a_2 b_1 + b_1 b_2)]$$

2) $N=16$ or operator in $GF(4^2)$

The elements in the subfield $GF(4)$ are: 0, 1, 2, 3. Addition and multiplication are defined by the Table 4.

Table 4

X \ Y	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Addition

X \ Y	0	1	2	3
0	0	0	0	0
1	1	0	1	2
2	2	0	2	3
3	3	0	3	1

Multiplication

$\delta=1$ is such that $\text{Trace}(\delta)=1$; then the product of two elements in $GF(16)$ is defined as above by operations of elements in $GF(4)$. Binary logic implementation of the mentioned operators in both $GF(4)$ and $GF(16)$ are shown in figures 4, 5.

Assume now that all the elements in $GF(4)$ are represented directly as a quaternary logic signal rather than by a pair of binary signals. Similarly an element β_1 in $GF(16)$, which are originally associated with a binary fourtuples can be represented also either by a 16 valued signal or a pair of quaternary signals. Then, if sum and product gates in $GF(4)$ are available directly, the similar gates in $GF(16)$ can be constructed from gates in the subfield as shown in Figure 6.

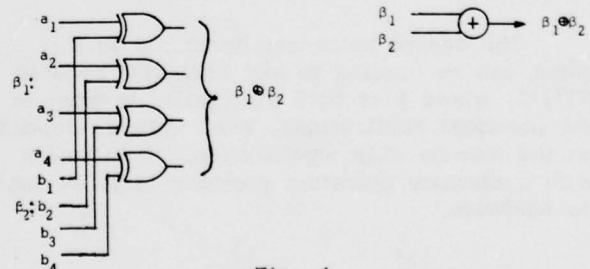


Fig. 4

Binary Implementation of Addition in $GF(16)$

It should be pointed out, that those operators in $GF(4)$ can be realized in current mode logic either in ECL or in integrated injection logic I^2L , where the logic signal takes four discrete well-defined current values (0,1,2I,3I).^{7,8}

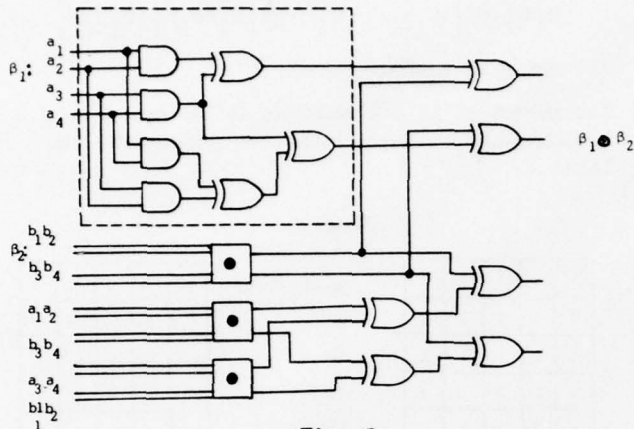


Fig. 5

Binary Implementation of Multiplication in $GF(16)$

CONCLUSION

We have discussed why a nonbinary SEC-DED code is suitable for a large memory which is byte organized. We deal specifically with $GF(2^2)$ and $GF(2^4)$, corresponding to bytes of size 2 and 4. A "Dynamic Programming" procedure for the design of an optimum parity matrix is sketched out. Quadratic extension field of the subfield is then introduced.

A complete design of an (80,64) SEC-DED code in $GF(2^4)$ is given. First the H matrix of the code and then the encoding-decoding implementation are optimized for maximum speed. Technique for realization of operators in the extended Galois field with binary and quaternary operators is outlined. Due to space limitation, description of I^2L operators in $GF(4)$ is not given.

The design procedure described in this paper can be applied to any SEC-DED code in $GF(2^p)$, where p or byte size takes in most of the practical applications, even values, depending on the memory chip organization. Realization with quaternary operators provides large savings on hardware.

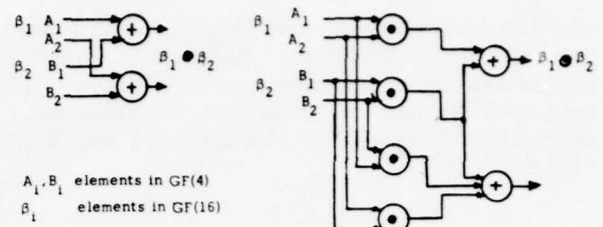


Fig. 6

Quaternary Implementation of Operators in $GF(16)$

REFERENCES

1. "A guide to the IBM 370 Model 145 GC 20-1734," IBM Corporation, 1970.
2. W.W. Peterson: "Error Correcting Codes," MIT Press, 1961, p. 52.
3. M.Y. Hsiao: "A class of optimal minimum odd-weight column SEC-DED codes," IBM M. Res. Dev., July 1970, pp.395-401.
4. D.C. Bossen: "b-Adjacent error correction," IBM J.Res.Dev., July 1970, pp. 402-408.
5. W. Ulrich: "Non binary error correction codes, BSTJ, November, 1957, pp. 1341.
6. R. Bellman: "Dynamic Programming," Princeton U. Press, 1957.
7. T.Tich Dao: "Threshold I^2L and its applications to binary symmetric functions and multivalued logic," IEEE J. of SSC., Oct., 1977, pp. 463-475.
8. T.Tich Dao, E.J. McCluskey, L.K. Russell: "Multivalued I^2L ," IEEE Transaction on Computers, Dec. 1977.
9. E. Belerkamp, Rumsey, Solomon: "On the solution of algebraic equations over finite fields," Information and Control, 1967, pp. 553-564.

ACKNOWLEDGEMENT

The author wishes to express his thanks to Ampex Corporation for allowing this work to be published and to Mrs. C. Gast for her careful editing and typing of the manuscript.

Tsutomu Sasao

Department of Electronic Engineering
Osaka University, Osaka 565, Japan

Abstract: A three-level programmable logic array (three-level PLA) consists of three main parts, the D array, the AND array, and the OR array, and each of these arrays can be programmed. In this paper, a design method for three-level PLA's is described. Main results obtained are 1) The minimization of the AND array corresponds to the minimization of a multiple-valued input two-valued output logic function; 2) By using the theory of multiple-valued decomposition of two-valued function, the computation time and the memory requirement for the minimization of the AND array can be reduced; and 3) The design of multiple-output function can be done in a similar way by introducing a variable which denotes the outputs.

I. Introduction

In the development of new integrated circuits, high cost as well as excessive lead time have long been recognized as serious problems by both the manufacturers and users of semiconductor devices. One approach to solve these problems that has appeared commercially over the years involves customizing only the interconnection pattern of standard prediffused array of logic gates. The other approach is now generally known as programmable logic arrays[1]-[5].

In this paper, a design method for three-level PLA's is described. The three-level PLA consists of three main parts, the D array, the AND array, and the OR array as shown in Fig.1.1, and each of these arrays can be programmed. For example, a six-input three-output function can be realized by the three-level PLA shown in Fig.1.2. In the D array, the horizontal lines are distributed OR gates with inputs represented as X's at selected line crossing. In the AND array, the dots are analogous to AND gate inputs where the gate is represented by the vertical line. In the OR array, the X's denote the OR gate inputs where the gate is represented by the horizontal line.

Three-level PLA's have several advantages to the conventional two-level PLA's[5].

- (1) In order to realize an arbitrary function

of n -variables, the array size of $O(2^n)$ is sufficient in a three-level PLA realization while $O(n2^n)$ is necessary in a conventional two-level PLA realization.

- (2) Array size can be further reduced by utilizing the partial symmetry, the decomposability, and the redundancy of the given function.

Major disadvantages of three-level PLA's are as follows:

- (3) Three-level PLA's are slower than two-level PLA's.

- (4) For small n , three-level PLA's sometimes require larger arrays than two-level PLA's.

In Section II, a design method for three-level PLA's which is obtained in [5] will be discussed. And it will be shown that the minimization of multiple-valued input two-valued output function corresponds to the minimization of the AND array. In Section III, the theory of multiple-valued decomposition of two-valued function will be introduced. And it will be shown that the computation time and memory requirement for the minimization of the AND array can be reduced by using the theory. In Section IV, a design method for multiple-output functions will be discussed.

II. Three-level Programmable Logic Arrays.

In this section, a design method which minimizes the size of a three-level PLA will be considered. As shown in Fig.1.1, the three-level PLA can realize an arbitrary OR-AND-OR circuits. Several works are known about OR-AND-OR circuits minimization[6]-[7], but these methods need too much computation even if the number of input variables is small. To avoid this difficulty, the design of three-level PLA is divided into two parts. The first part is the design of the D array, and the second part is the design of the AND array. It will be shown that in order to minimize the size of the AND array for a given function, it is sufficient to obtain a minimal sum-of-products expression for the corresponding multiple-valued input two-valued output function.

Definition 2.1: A three-level PLA consists of the D array, the AND array, and the OR array as shown in Fig.1.1. The size of n -variable m -output three-level PLA is defined as $C(n) = (2n+W)H+Wm$, where W is the number of columns of the AND array, H is the number of rows of the AND array, and m is the number of rows of the OR array.

Definition 2.2: Let $X = (x_1, x_2, \dots, x_n)$ be a variable in $B^n = \{0, 1\}^n$. The set of variables in X is denoted by $\{x_1, x_2, \dots, x_n\}$ or by $\{X\}$. The number of the variables in $\{X\}$ is denoted by $d(X)$. (X_1, X_2, \dots, X_r) is said to be a partition of X iff $\{X_1\} \cup \{X_2\} \cup \dots \cup \{X_r\} = \{X\}$, $\{X_i\} \cap \{X_j\} = \emptyset$ ($i \neq j$), and $\{X_i\} \neq \emptyset$.

Definition 2.3: Let $a = (a_1, a_2, \dots, a_n)$ be a constant in B^n . $X^a: B^n \rightarrow B$ is a function such that $X^a = 0$ if $X \neq a$ and $X^a = 1$ if $X = a$. Let $S \subseteq B^n$, X^S is defined as $X^S = \bigvee_{a \in S} X^a$.

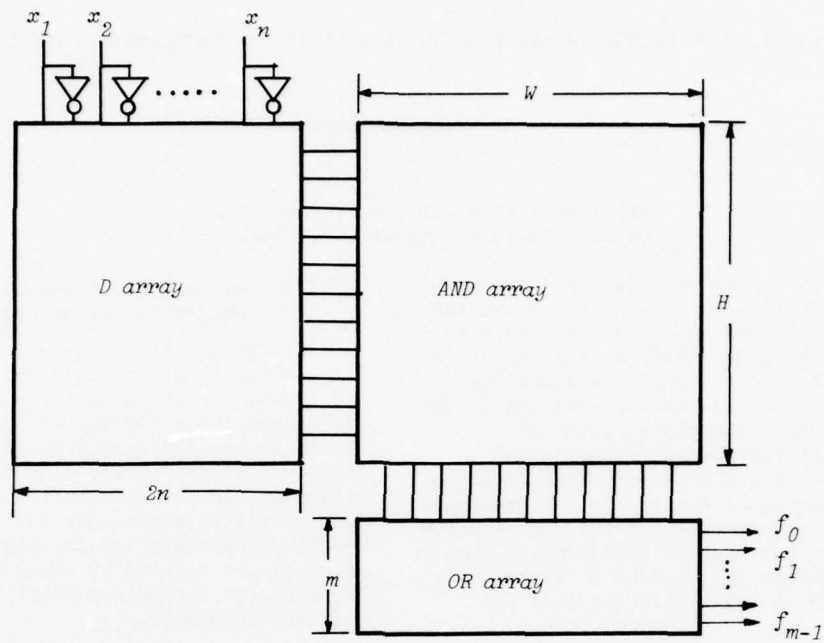


Fig. 1.1 Three-level PLA

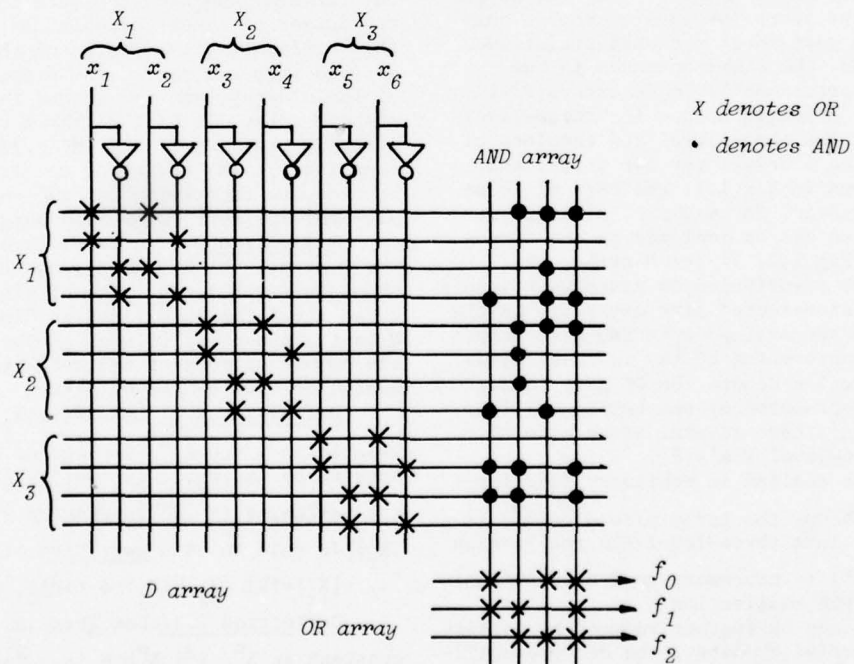


Fig.1.2 An example of three-level PLA

Lemma 2.1: Let (X_1, X_2, \dots, X_r) be a partition of X . An arbitrary function $f(X)$ of n variable is expressed in the form

$$\bigvee_{(a_1, a_2, \dots, a_r)} f(a_1, a_2, \dots, a_r) \bar{x}_1^{a_1} \bar{x}_2^{a_2} \dots \bar{x}_r^{a_r},$$

where $a_i \in B^{n_i}$, $d(X)=n$, and $d(X_i)=n_i$.

Example 2.1: Consider the four-variable function

$f(X) = x_1 x_2 \bar{x}_3 \bar{x}_4 \vee x_1 \bar{x}_2 x_3 x_4 \vee x_1 \bar{x}_2 x_3 \bar{x}_4 \vee \bar{x}_1 x_2 x_3 x_4$.
Let (X_1, X_2) be a partition of $X = (x_1, x_2, x_3, x_4)$ and let $X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$. $f(X)$ can be represented in the form

$$f(X) = x_1^{(11)} \bar{x}_2^{(00)} \vee x_1^{(11)} \bar{x}_2^{(01)} \vee x_1^{(10)} \bar{x}_2^{(11)} \vee x_1^{(01)} \bar{x}_2^{(11)}.$$

Lemma 2.2: Let $d(X)=n$ and $S_1, S_2 \subseteq B^n = I$.

$$\frac{S_1}{X} \cdot \frac{S_2}{X} = \frac{S_1 \cap S_2}{X}, \quad \frac{S_1}{X} \vee \frac{S_2}{X} = \frac{S_1 \cup S_2}{X},$$

$$\frac{S_1}{X} = \frac{I - S_1}{X}, \quad X^I = 1, \text{ and } X^\emptyset = 0.$$

Definition 2.4: X^S is said to be a literal. A product of distinct literals is said to be a term. A sum of terms is said to be a sum-of-products expression. The number of terms in a sum-of-products expression P is denoted by $t(P)$. P is said to be minimal if there is no expression Q such that $t(Q) < t(P)$ and that Q denotes the same function as P . Let E_1 and E_2 be terms. E_2 is subterm of E_1 iff $E_1 \neq E_2$ and $E_1 \leq E_2$. E_1 is said to be a prime implicant of f if $E_1 \leq f$ and if there is no subterm E_2 of E_1 such that $E_2 \leq f$.

Lemma 2.3: Let (X_1, X_2, \dots, X_r) be a partition of X . An arbitrary function $f(X)$ can be represented in a form

$$f(X) = \bigvee_{(S_1, S_2, \dots, S_r)} \bar{x}_1^{S_1} \bar{x}_2^{S_2} \dots \bar{x}_r^{S_r}, \quad (2.1)$$

where $S_i \subseteq B^{n_i}$, and $d(X_i)=n_i$. In a three-level PLA, if the D array generates all the maxterms of $\{X_i\}$ for $i=1, 2, \dots, r$, then an arbitrary term which has the form $\bar{x}_1^{S_1} \bar{x}_2^{S_2} \dots \bar{x}_r^{S_r}$ can be realized in each column of the AND array.

Example 2.2: The function of Example 2.1 can be represented as

$$f(X) = x_1^{(11)} \bar{x}_2^{\{(00), (01)\}} \vee x_1^{\{(10), (01)\}} \bar{x}_2^{(11)}.$$

Theorem 2.1: Let (X_1, X_2, \dots, X_r) be a partition of X , and let the D array generate all the maxterms of $\{X_i\}$ for $i=1, 2, \dots, r$. In order to minimize the size of the AND array for $f(X)$, it is sufficient to obtain a minimal sum-of-products expression of $f(X)$ having the form

$$f(X_1, X_2, \dots, X_r) = \bigvee_{(S_1, S_2, \dots, S_r)} \bar{x}_1^{S_1} \bar{x}_2^{S_2} \dots \bar{x}_r^{S_r}.$$

If P is minimal expression for $f(X)$, then $t(P) \leq 2^{n - \max\{n_i\}}$, where $n_i = d(X_i)$.

Proof: By Lemma 2.3, we have the first part. Assume without loss of generality that $n_1 = \max\{n_i\}$. $f(X)$ can be represented as

$$f(X_1, X_2, \dots, X_r) = \bigvee_{(S_1, a_2, a_3, \dots, a_r)} \bar{x}_1^{S_1} \bar{x}_2^{a_2} \bar{x}_3^{a_3} \dots \bar{x}_r^{a_r} \quad (2.2)$$

The number of terms in (2.2) is at most

$$\prod_{i=2}^r 2^{n_i} = 2^{n - n_1} = 2^{n - \max\{n_i\}}. \quad \text{Q.E.D.}$$

Corollary 2.1: The size of three-level PLA which is sufficient to realize an arbitrary function of n variable is $C(n) = (2n+W)H+W$, where

$$W = 2^{n - \max\{n_i\}}, \quad H = \sum_{i=1}^r 2^{n_i} \quad \text{and } \underline{n} = (n_1, n_2, \dots, n_r) \text{ is a}$$

vector which represents the partition of input variables.

III. Multiple-Valued Decomposition of

Two-Valued Logic Function.

In this section, the theory of multiple-valued decomposition of two-valued function will be described. By using this theory, we can reduce the computation time and the memory requirement for the minimization of the AND arrays.

Definition 3.1: Let (X_1, X_2, \dots, X_r) be a partition of X , and $f(X)$ be a function such that

$$f: B^{n_1} \times B^{n_2} \times \dots \times B^{n_r} \rightarrow B.$$

For $a, b \in B^{n_i}$, define a relation

$$a \stackrel{i}{\sim} b \Leftrightarrow f(X|a \rightarrow X_i) = f(X|b \rightarrow X_i),$$

where $f(X|a \rightarrow X_i)$ denotes $f(X_1, X_2, \dots, X_{i-1}, a, X_{i+1}, \dots, X_r)$. Obviously, the relation $\stackrel{i}{\sim}$ is an equivalence

relation. Let $\Pi_i = (L_0^i, L_1^i, \dots, L_{k_i-1}^i)$ be a partition of B^{n_i} induced by the equivalence relation $\stackrel{i}{\sim}$.

A function $\psi_i: B^{n_i} \rightarrow M_i$; $M_i = \{0, 1, \dots, k_i-1\}$ such that $\psi_i(a) = j \Leftrightarrow a \in L_j^i$ is called a partition function of B^{n_i} .

Example 3.1: Consider a six-variable function

$$f(X) = (\bar{x}_1 \vee \bar{x}_2) \cdot (x_3 \oplus x_4) \cdot (\bar{x}_5 \vee \bar{x}_6) \vee (x_1 \vee x_2) \cdot (x_3 \oplus \bar{x}_4) x_5 \vee (x_1 \oplus x_2) \cdot (x_5 \oplus x_6).$$

Let (X_1, X_2, X_3) be a partition of X , where $X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$, and $X_3 = (x_5, x_6)$. Note that

$$f(X|(01) \rightarrow X_1) = f(X|(10) \rightarrow X_1),$$

$$f(X|(00) \rightarrow X_2) = f(X|(11) \rightarrow X_2), \text{ and}$$

$$f(X|(10) \rightarrow X_2) = f(X|(01) \rightarrow X_2).$$

The partition functions of B^{n_i} are shown in Table 3.1.

Table 3.1 Partition functions

X_i	$\psi_1(X_1)$	$\psi_2(X_2)$	$\psi_3(X_3)$
00	0	0	0
01	1	1	1
10	1	1	2
11	2	0	3

Lemma 3.1: Let (X_1, X_2, \dots, X_r) be a partition of X , $d(X_i) = n_i$, and let ψ_i be a partition function of B^{n_i} . There exists a multiple-valued input two-valued output function

$g: M_1 \times M_2 \times \dots \times M_r \rightarrow B$ such that
 $f(X_1, X_2, \dots, X_r) = g(\psi_1(X_1), \psi_2(X_2), \dots, \psi_r(X_r))$.

Proof: If $a_i \in L_{b_i}^1$ ($i=1, 2, \dots, r$), then let
 $g(b_1, b_2, \dots, b_r) = f(a_1, a_2, \dots, a_r)$. It is easy to show that this function satisfies the condition. Q.E.D.

This lemma is similar to the well known decomposition theorem of Ashenhurst[8]-[11]. But ψ_i is, in general, a multiple-valued function. When $M_i = \{0, 1\}$ ($i=1, 2, \dots, r$), this lemma reduced to the ordinary decomposition theorem.

Example 3.2: Consider the function of Example 3.1. By Lemma 3.1, $f(X)$ can be represented as $f(X_1, X_2, X_3) = g(\psi_1(X_1), \psi_2(X_2), \psi_3(X_3))$, where $g(Y_1, Y_2, Y_3)$ is shown in Table 3.2.

Table 3.2

Y_1	Y_2	Y_3	g
0	0	0	0
0	0	1	0
0	0	2	0
0	0	3	0
0	1	0	1
0	1	1	1
0	1	2	1
0	1	3	0
1	0	0	0
1	0	1	1
1	0	2	1
1	0	3	1
1	1	0	1
1	1	1	1
1	1	2	1
1	1	3	0
2	0	0	0
2	0	1	0
2	0	2	1
2	0	3	1
2	1	0	0
2	1	1	0
2	1	2	0
2	1	3	0

Definition 3.2: Let $M = \{0, 1, \dots, k-1\}$, $t \in M$, and $Y^t: M \rightarrow B$ be a function such that $Y^t = 0$ if $Y \neq t$ and $Y^t = 1$ if $Y = t$. Let $T \subseteq M$, Y^T is a function such that $Y^T = \bigvee_{t \in T} Y^t$.

Lemma 3.2: Let $T_1, T_2 \subseteq M = I$.

$$Y^{T_1} \cdot Y^{T_2} = Y^{T_1 \cap T_2}, \quad Y^{T_1} \vee Y^{T_2} = Y^{T_1 \cup T_2},$$

$$Y^{I-T_1} = Y^{I-T_1}, \quad Y^{I-T_1} = 1, \quad \text{and } Y^\emptyset = 0.$$

Lemma 3.3: A multiple-valued input two-valued output function

$$g: M_1 \times M_2 \times \dots \times M_r \rightarrow B$$

can be represented in the form

$$g(Y_1, Y_2, \dots, Y_r) =$$

$$\bigvee_{(t_1, t_2, \dots, t_r)} g(t_1, t_2, \dots, t_r) Y_1^{t_1} Y_2^{t_2} \dots Y_r^{t_r} \quad \text{----- (3.1)}$$

or in a form

$$g(Y_1, Y_2, \dots, Y_r) = \bigvee_{(T_1, T_2, \dots, T_r)} Y_1^{T_1} Y_2^{T_2} \dots Y_r^{T_r}, \quad \text{----- (3.2)}$$

where $t_i \in T_i$, $T_i \subseteq M_i$, and $M_i = \{0, 1, \dots, k_i - 1\}$.

Proof: By Definition 3.2, it is easy to show that (3.1) holds. By Lemma 3.2 and (3.1), we have (3.2). Q.E.D.

Example 3.3: The function g of Example 3.2 can be represented in the form

$$g(Y_1, Y_2, Y_3) = Y_1^0 Y_2^1 Y_3^0 \vee Y_1^0 Y_2^1 Y_3^1 \vee Y_1^0 Y_2^1 Y_3^2 \vee Y_1^1 Y_2^0 Y_3^1 \vee Y_1^1 Y_2^0 Y_3^2 \vee Y_1^1 Y_2^0 Y_3^3 \vee Y_1^1 Y_2^1 Y_3^0 \vee Y_1^1 Y_2^1 Y_3^1 \vee Y_1^1 Y_2^1 Y_3^2 \vee Y_1^2 Y_2^0 Y_3^2 \vee Y_1^2 Y_2^0 Y_3^3, \quad \text{----- (3.3)}$$

or in a form

$$g(Y_1, Y_2, Y_3) = Y_1^{\{0,1\}} \cdot Y_2^{\{0,1,2\}} \vee Y_1^{\{1,2\}} \cdot Y_2^{\{0\}} \cdot Y_3^{\{2,3\}} \vee Y_1^{\{1\}} \cdot Y_3^{\{1,2\}} \quad \text{----- (3.4)}$$

By using positional cube notations to represent terms [12]-[14], (3.3) and (3.4) can be represented as Table 3.3 and Table 3.4, respectively.

Table 3.3

Y_1	Y_2	Y_3
012	01	0123
100-01-1000		
100-01-0100		
100-01-0010		
010-10-0100		
010-10-0010		
010-10-0001		
010-01-1000		
010-01-0100		
010-01-0010		
001-10-0010		
001-10-0001		

Table 3.4

Y_1	Y_2	Y_3
012	01	0123
110-01-1110		
011-10-0011		
010-11-0110		

Lemma 3.4: Let f, ψ_1 , and g be functions such that

$$f: B_1^{n_1} \times B_2^{n_2} \times \dots \times B_r^{n_r} \rightarrow B,$$

$$\psi_i: B_1^{n_i} \rightarrow M_i; \quad M_i = \{0, 1, \dots, k_i - 1\},$$

$$g: M_1 \times M_2 \times \dots \times M_r \rightarrow B.$$

and $f(X_1, X_2, \dots, X_r) = g(\psi_1(X_1), \psi_2(X_2), \dots, \psi_r(X_r))$.

Let $\psi_i = (L_0^i, L_1^i, \dots, L_{k_i-1}^i)$ be a partition function of B^{n_i} induced by the relation $\stackrel{i}{\sim}$, and let

$$\psi_i(a) = j \iff a \in L_j^i. \text{ A literal } Y_i^j \text{ of the expression } g(Y_1, Y_2, \dots, Y_r) \text{ corresponds to a literal } X_i^{S_i}; \quad S_i = \bigcup_{j \in T_i} L_j^i$$

of the expression $f(X_1, X_2, \dots, X_r)$. And a term $Y_1^{T_1} \cdot Y_2^{T_2} \cdot \dots \cdot Y_r^{T_r}$ of $g(Y)$ corresponds to a term $X_1^{S_1} \cdot X_2^{S_2} \cdot \dots \cdot X_r^{S_r}$ of $f(X)$.

Proof: It is easy to show by Definition 3.1 and Definition 3.2. Q.E.D.

Example 3.4: Consider the function $f(X)$ of Example 3.1 and the function $g(Y)$ of Example 3.3. For the term $Y_1^{0,1} \cdot Y_2^{1,2}$ of $g(Y)$, the corresponding term of $f(X)$ is $X_1^{\{(00), (01), (10)\}} \cdot X_2^{\{(01), (10)\}} \cdot X_3^{\{(00), (01), (10)\}}$. For the term $Y_1^{1,2} \cdot Y_2^{0,1} \cdot Y_3^{2,3}$, the corresponding term of $f(X)$ is $X_1^{\{(01), (10), (11)\}} \cdot X_2^{\{(00), (11)\}} \cdot X_3^{\{(10), (11)\}}$, and for the term $Y_1^{1,2} \cdot Y_3^{1,2}$, the corresponding term of $f(X)$ is $X_1^{\{(01), (10)\}} \cdot X_3^{\{(01), (10)\}}$.

Theorem 3.1: Let two expressions

$$f(X_1, X_2, \dots, X_r) = \bigvee_{(S_1, S_2, \dots, S_r)} X_1^{S_1} \cdot X_2^{S_2} \cdot \dots \cdot X_r^{S_r}$$

and

$$g(Y_1, Y_2, \dots, Y_r) = \bigvee_{(T_1, T_2, \dots, T_r)} Y_1^{T_1} \cdot Y_2^{T_2} \cdot \dots \cdot Y_r^{T_r}$$

satisfy the relation $f(X_1, X_2, \dots, X_r) = g(\psi_1(X_1), \psi_2(X_2), \dots, \psi_r(X_r))$. If P_1 and Q_1 are minimal expressions for $f(X)$ and $g(Y)$, respectively, then

$$t(P_1) = t(Q_1) \leq \left(\prod_{i=1}^r k_i \right) / (\max\{k_i\}),$$

where $\psi_i: B \rightarrow M_i; \quad M_i = \{0, 1, \dots, k_i - 1\}$, and $d(X_i) = n_i$.

Proof: (1) For P_1 , a minimal sum-of-products expression of $f(X)$, consider the expression P_2 which has the form

$$\bigvee_{(G_1, G_2, \dots, G_r)} Y_1^{G_1} \cdot Y_2^{G_2} \cdot \dots \cdot Y_r^{G_r}, \text{ where}$$

$G_i = \{j | L_j^i \subseteq A_i\}$ and $A_i = \{a | a \stackrel{i}{\sim} b, b \in S_i\}$. Clearly, $t(P_1) = t(P_2)$. It is easy to show that P_2 represents $g(Y)$. For Q_1 , a minimal sum-of-products expression of $g(Y)$, consider the expression Q_2 which has the form

$$\bigvee_{(D_1, D_2, \dots, D_r)} X_1^{D_1} \cdot X_2^{D_2} \cdot \dots \cdot X_r^{D_r}, \text{ where } D_i = \bigcup_{j \in T_i} L_j^i.$$

Clearly, $t(Q_1) = t(Q_2)$. By Lemma 3.4, Q_2 represents $f(X)$. As P_1 is a minimal expression of $f(X)$, we have $t(P_1) \leq t(Q_2)$. As Q_1 is a minimal expression of $g(Y)$, we have $t(Q_1) \leq t(P_2)$. Therefore, $t(P_1) = t(Q_1)$.

(2) Assume without loss of generality that $\max\{k_i\} = k_1$. $g(Y)$ can be represented in the form

$$g(Y_1, Y_2, \dots, Y_r) = \bigvee_{(T_1, T_2, \dots, T_r)} Y_1^{T_1} \cdot Y_2^{T_2} \cdot Y_3^{T_3} \cdot \dots \cdot Y_r^{T_r}, \quad \text{----- (3.5)}$$

where $T_1 \subseteq M_1$ and $t_i \in M_i$ ($i=2, 3, \dots, r$). The number of terms in (3.5) is at most

$$\prod_{i=2}^r k_i = \left(\prod_{i=1}^r k_i \right) / (\max\{k_i\}).$$

Hence, we have the theorem. Q.E.D.

Theorem 3.2: Let (X_1, X_2, \dots, X_r) be a partition

of X , and the D array generates all the maxterms of X_i for $i=1, 2, \dots, r$. In order to minimize the size of the AND array for

$$f(X_1, X_2, \dots, X_r) = g(\psi_1(X_1), \psi_2(X_2), \dots, \psi_r(X_r)),$$

it is sufficient to obtain a minimal sum-of-products expression of $g(Y)$ having the form

$$g(Y_1, Y_2, \dots, Y_r) = \bigvee_{(T_1, T_2, \dots, T_r)} Y_1^{T_1} \cdot Y_2^{T_2} \cdot \dots \cdot Y_r^{T_r}.$$

Proof: By Theorem 3.1. Q.E.D.

Example 3.5: The expression (3.4) of Example 3.3 is a minimal sum-of-products expression for $g(Y)$. Therefore, the corresponding minimal expression for $f(X)$ is given by $f(X_1, X_2, X_3) =$

$$X_1^{\{(00), (01), (10)\}} \cdot X_2^{\{(01), (10)\}} \cdot X_3^{\{(00), (01), (10)\}} \vee X_1^{\{(01), (10), (11)\}} \cdot X_2^{\{(00), (11)\}} \cdot X_3^{\{(10), (11)\}} \vee X_1^{\{(01), (10)\}} \cdot X_3^{\{(01), (10)\}}.$$

Table 3.5 shows the positional cube notation for $f(X)$.

Table 3.5 Positional cubes for f

X ₁				X ₂				X ₃			
00	01	10	11	00	01	10	11	00	01	10	11
1	1	1	0	0	1	1	0	1	1	1	0
0	1	1	1	0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	1	0	1	1	0

Corollary 3.1: Let $f(X)$ be a function such that

$$f(X_1, X_2, \dots, X_r) = g(\psi_1(X_1), \psi_2(X_2), \dots, \psi_r(X_r)).$$

 The size of three-level PLA which is sufficient to realize $f(X)$ is given by $C(n) = (2n+W)H+W$, where

$$W = \left(\prod_{i=1}^r k_i \right) / (\max\{k_i\}), \quad H = \sum_{i=1}^r 2^{n_i}, \quad \text{and } \psi_i: B^{n_i} \rightarrow M_i;$$

 $M_i = \{0, 1, \dots, k_i - 1\}.$

As shown in the examples of this section, it is clear that obtaining a minimal sum-of-products expression for $g(Y)$ requires less computation time and less memories than obtaining that for $f(X)$. The multiple-valued decomposition of two-valued logic function can be done in a similar way to ordinary two-valued decomposition.

IV. Synthesis of Multiple-Output Functions.

In the case of multiple-output functions, simultaneous minimization often produces better solution than individual minimization [15], [16]. In this section, we will show that the minimization of AND array for a multiple-output function as well as a single-output function can be done by a minimization of a multiple-valued input two-valued output function.

Theorem 4.1: In order to minimize the AND array for m output functions

$$f_j: B^{n_1} \times B^{n_2} \times \dots \times B^{n_r} \rightarrow B \quad (j=0, 1, \dots, m-1),$$

it is sufficient to obtain a minimal sum-of-products expression for the function

$$F: B^{n_1} \times B^{n_2} \times \dots \times B^{n_r} \times M \rightarrow B$$

having a form

$$F(X_1, X_2, \dots, X_r, Z) = \bigvee_{(S_1, S_2, \dots, S_r) \in R} S_1^{S_1} \cdot S_2^{S_2} \cdot \dots \cdot S_r^{S_r} \cdot Z^R,$$

where $F(X_1, X_2, \dots, X_r, j) = f_j(X_1, X_2, \dots, X_r)$, $S_i \in B^{n_i}$, $R \subseteq M$, and $M = \{0, 1, \dots, m-1\}$.

Proof: For expressions of f_j ($j=0, 1, \dots, m-1$):

$$f_j(X_1, X_2, \dots, X_r) = \bigvee_{(S_1, S_2, \dots, S_r) \in R} S_1^{S_1} \cdot S_2^{S_2} \cdot \dots \cdot S_r^{S_r},$$

consider a expression for F shown above.

By definition,

$$S_1^{S_1} \cdot S_2^{S_2} \cdot \dots \cdot S_r^{S_r} \leq f_j \Leftrightarrow S_1^{S_1} \cdot S_2^{S_2} \cdot \dots \cdot S_r^{S_r} \cdot Z^R \leq F$$

and $j \in R$.

It is easy to see that the number of terms in F is equal to the number of columns of the AND array. Hence the theorem. Q.E.D.

Example 4.1: Consider the six-variable three-output function shown in Table 4.1, where the function is represented as a set of positional cubes. Let the partition of X be (X_1, X_2, X_3) , where

$X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$, and $X_3 = (x_5, x_6)$. Let Z be a variable which denotes the outputs. Consider the function

$$F(X_1, X_2, X_3, Z): B^2 \times B^2 \times B^2 \times \{0, 1, 2\} \rightarrow B.$$

F has the following properties:

$$F(X_1, (01), X_3, Z) = F(X_1, (10), X_3, Z);$$

$$F(X_1, X_2, (00), Z) = F(X_1, X_2, (11), Z); \text{ and}$$

$$F(X_1, X_2, (10), Z) = F(X_1, X_2, (01), Z).$$

So F can be decomposed as

$$F(X_1, X_2, X_3, Z) = G(\psi_1(X_1), \psi_2(X_2), \psi_3(X_3), Z),$$

where ψ_i are shown in Table 4.2. G is a function such that

$$G: \{0, 1, 2, 3\} \times \{0, 1, 2\} \times \{0, 1\} \times \{0, 1, 2\} \rightarrow \{0, 1\}.$$

By Theorem 4.1, in order to minimize the size of the AND array for the multiple-output function, it is sufficient to obtain a minimal sum-of-products expression for G . The terms of G are shown in Table 4.3. It is easy to see that this is a minimal sum-of-products expression for G . The minimal sum-of-products expression for F which corresponds to G is shown in Table 4.4. The three-level PLA which realizes the given functions is shown in Fig. 1.2.

Table 4.1 Six-variable three-output function

Input						Output		
x_1	x_2	x_3	x_4	x_5	x_6	f_0	f_1	f_2
01	01	01	01	01	01	1	1	0
11	10	11	10	10	10	1	1	0
11	10	11	10	01	01	1	1	0
11	10	10	11	10	10	1	1	0
11	10	10	11	01	01	1	1	0
10	11	11	10	10	10	1	1	0
10	11	11	10	01	01	1	1	0
10	11	10	11	10	10	1	1	0
10	11	10	11	01	01	1	1	0
01	11	01	01	10	10	0	1	1
01	11	01	01	01	01	0	1	1
11	01	01	01	10	10	0	1	1
11	01	01	01	01	01	0	1	1
10	01	10	01	11	11	1	0	1
10	01	10	01	11	11	1	0	1
01	10	01	11	10	10	1	1	1
01	10	01	11	01	01	1	1	1
01	10	11	01	10	10	1	1	1
01	10	11	01	01	01	1	1	1
10	01	01	11	10	10	1	1	1
10	01	01	11	01	01	1	1	1
10	01	11	01	10	10	1	1	1
10	01	11	01	01	01	1	1	1

- [17] D.L.Dietmeyer and P.R.Schneider, "Identification of symmetry, redundancy, and equivalence class of Boolean functions," IEEE Trans. Electron Comput. vol.EC-16, pp.804-817, Dec. 1967.
- [18] S.C.Lee, "Vector Boolean algebra and calculus," IEEE Trans. Comput. vol. C-25, pp.865-874, Sept. 1976.
- [19] J.Dussault, G.Metze and M.Krieger, "A multi-valued switching algebra with Boolean properties," Proc. of 1976 Inter. Sympo. on Multiple-Valued Logic, pp.68-73, May 1976.
- [20] P.T.Cheung, "Identification of different functional properties of multiple valued Switching functions," Proc. of 1976 Inter. Sympo. on Multiple-Valued Logic, pp.74-78, May 1976.
- [21] R.M.Karp, "Reducibility among combinatorial problems," Complexity of Computer Computations, R.E.Miller and J.W. Thatcher. eds. Plenum Press, New York, 1972.
- [22] Y.Kambayashi, "Optimum logic design using memory-type arrays," Proceedings of the International Symposium on Uniformly Structured Automata and Logic, Tokyo, Aug. 1975.
- [23] J.P.Deschamps and A.Thayse, "Representation of discrete functions " Proc. of 1975 Inter. Symposium on Multiple-Valued Logic, pp.99-111, May 1975.
- [24] A.Thayse and J.P.Deschamps, "Logic properties of unate discrete and switching functions," IEEE Trans. Comput. vol. C-26, pp.1202-1212, Dec. 1977.
- [25] J.P.Roth, "Programmed logic array optimization," IEEE Trans. Comput. vol. C-27, pp.174-176, February 1978.

A SIMULTANEOUS ANALOG / TERNARY CONVERTER

M. A. H. Abdul - Karim and N. E. Berbat *

* Department of Electrical Engineering , College of Engineering ,
University of Baghdad , Baghdad , IRAQ .

This note presents a technique of designing 2 bit simultaneous analog to ternary convertor with 100 ns response time . TTL logic components were used throughout the prototype design .

$$2 \equiv 3 \text{ volt} \geq v \geq 1.5 \text{ volt}$$

$$1 \equiv \geq 3 \text{ volt}$$

$$0 \equiv \leq 1.5 \text{ volt}$$

1 Introduction :

Recent technical work have shown some advantages of using multi - valued logic where the natural question was if there is a practical radix other than 2 that will produce circuits with greater saving in components , without loss of speed . Alexander (1964) showed that the most efficient radix for implementation of switching systems is the natural base ($e = 2.71828$) , it seems likely that the best integral radix is 3 rather than 2 . Ternary gives the meaning of a switching system which will perform 3 - valued transmission and 3 valued switching . A ternary variable Q will assume three values Q , \bar{Q} , $\bar{\bar{Q}}$, where the dash sign means cycling . This note deals with a circuitry which is able to perform a 2 bit A/T conversion that may be used as part of a instrumentation system , for example .

2 Circuit Working Principle

Figure -1- shows the block diagram of A/T convertor , and table -1 - gives truth table for this circuit .

Three distinct sections can be identified . The first is comprised of $3^n - 1$ analog comparators yielding $3^n - 1$ parallel signals . The second one is the binary encoder logic circuit which produces from $3^n - 1$ binary inputs $2n$ binary outputs . The third section is a transistor circuits designed to translate the binary outputs into ternary form . The logic states are represented by voltage levels as given below .

Table 1

V_{in}	Ternary o/p	
	$x = 3^0$	$y = 3^1$
0	0	0
1	1	0
2	2	0
3	0	1
4	1	1
5	2	1
6	0	2
7	1	2
8	2	2

Figure -2- illustrates a parallel 2 - bit A/T convertor which has been constructed and tested . It employs commercially available integrated circuits and discrete components throughout . The outputs of the comparators A , B , C , D , E , F , G and H are either at logic "0" or logic "1", and they are fed to the binary encoder logic circuit as shown in table -2- .

Table 2

V_{in}	Comparator Output								Encoder Logic o/p			
	A	B	C	D	E	F	G	H	S	T	U	V
0	0	0	0	0	0	0	0	0	0	1	0	1
0.5	1	0	0	0	0	0	0	0	0	0	0	1
1	1	1	0	0	0	0	0	0	1	0	0	1
1.5	1	1	1	0	0	0	0	0	0	1	0	0
2	1	1	1	1	0	0	0	0	0	0	0	0
2.5	1	1	1	1	1	0	0	0	1	0	0	0
3	1	1	1	1	1	1	0	0	0	1	1	0
3.5	1	1	1	1	1	1	1	0	0	0	1	0
4	1	1	1	1	1	1	1	1	1	0	1	0

The four binary encoder outputs S, T, U and V are also at either logic '0' or logic '1' and are applied to their respective output transistor circuit. The binary input data are translated into ternary form by means of the binary / ternary convertor circuit. This translating circuit is comprised of four transistors for a two bit ATC. When S and T are low, the output X will be at logic 1, i.e. $X \equiv 1$, but when S is high and T is low, $X \equiv 2$, however if S is low and T is high, $X \equiv 0$ as was defined before. The same reasoning applies for ternary output Y in terms of binary inputs U and V as shown in table -3-.

V _{in} Volts	Encoder Logic Output				Ternary o/p	
	S	T	U	V	X	y
0	0	1	0	1	0	0
0.5	0	0	0	1	1	0
1	1	0	0	1	2	0
1.5	0	1	0	0	0	1
2	0	0	0	0	1	1
2.5	1	0	0	0	2	1
3	0	1	1	0	0	2
3.5	0	0	1	0	1	2
4	1	0	1	0	2	2

Table - 3 -

The response time is determined by the input to output transfer delay of the comparators plus the worst case propagation delay through the binary encoder and the output transistor circuit. The comparators have a response time of around 40 ns. The binary encoder is comprised of three parts of 7400 series TTL logic; this and the translating binary / ternary transistor circuit will give together a propagation delay of around 60 ns. Thus the A/T convertor has a response time of around 100 ns, this may be improved by using higher speed components. The problem of the exponential increase, $O(3)^n$ in the necessary number of devices remains. Thus the complexity of simultaneous convertors is unavoidable.

References :

1. Alexander W. Feb. 1964 Electronics and Power 36.
2. Contoni A. July 1973 Proc. of IRE, 230.
3. Yoeli M. and Halpern I. 1968 Proc. IEE, 115, 10.

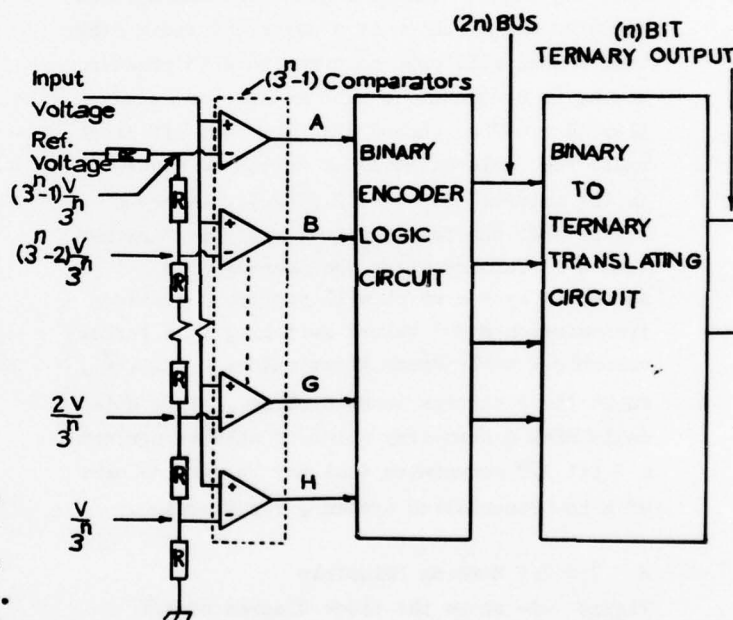


FIG1 Block Diagram of Simultaneous Analog-To-Ternary Converter

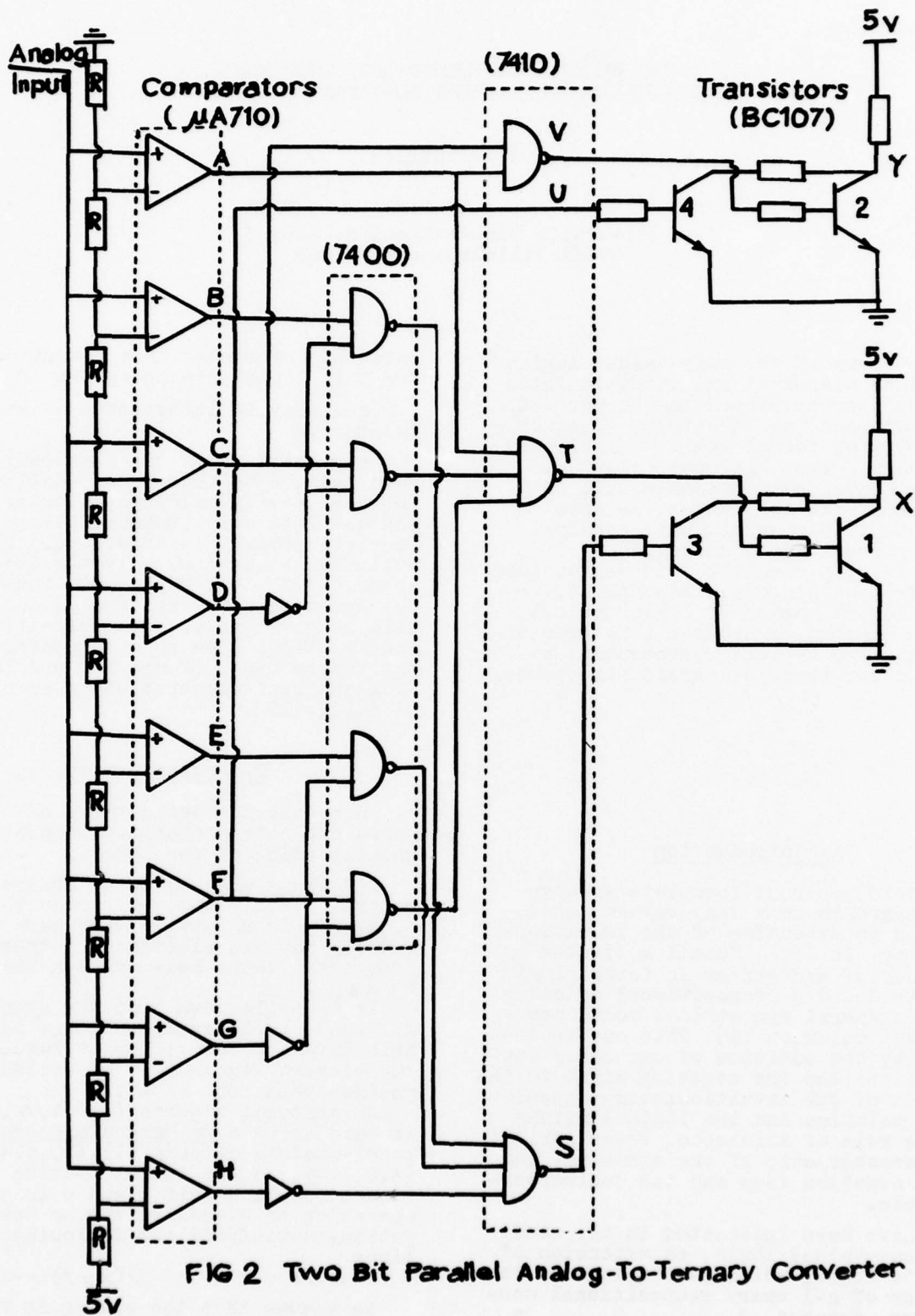


FIG 2 Two Bit Parallel Analog-To-Ternary Converter

TWO TYPICAL REPRESENTATION THEOREMS FOR SYMMETRICAL HEYTING ALGEBRAS OF ORDER n

Luisa Iturrioz

Université Claude-Bernard, Lyon I
69621 Villeurbanne, France

The theory of the many-valued logics related to classical and intuitionistic ones has been developed in the past. On the other hand, as an attempt to study symmetries on formal logic, Moisil has introduced a propositional calculus, called general symmetrical modal. In connection with the latter, we have built up, in a standard way, a many-valued propositional calculus.

In order to consider this many-valued logic from an algebraic standpoint, we introduce the notion of a symmetrical Heyting algebra of order n . We present here only two typical representation theorems for these algebraic structures.

rules of inference. In a semantical model for this logic, the operators S_i , $i=1, \dots, n-1$ may be interpreted as modal functors.

We present below the algebraic structure related to that logic -called a symmetrical Heyting algebra of order n -. The complete work (subdirect irreducibles free, injective algebras, etc.) will be published elsewhere. Only the discussion about two typical representation theorems -by means of sets and topological type- will be given here. Some definitions and constructions have been suggested by the particular cases of $n=3$ [4] and Łukasiewicz and Post algebras of order n , [11], [13], [2], [10].

2. Preliminaries

We recall the definitions of structures and some properties needed for the understanding of the paper.

1. Introduction

In this paper we formulate an algebraic approach to a many-valued logic, which is an extension of the intuitionistic one. In 1942, Moisil motivated by the study of symmetries in formal logic has introduced a propositional calculus -called general symmetrical modal propositional calculus- [6]. This one is obtained by the addition of one unary connective -called the negation sign- to the alphabet of the intuitionistic propositional calculus and two logical axioms and one rule of inference. These exhibit the characteristic of the symmetry: the double negation laws and the contraposition rule.

We have been interested in the study of a many-valued logic, an extension of Moisil's logic, which is obtained by the addition of $n-1$ unary propositional connectives, denoted S_1, S_2, \dots, S_{n-1} , n an integer ≥ 2 , with adequate axioms and

A Heyting algebra is an abstract algebra $(A, 0, 1, \wedge, \vee, \Rightarrow)$ such that $(A, 0, 1, \wedge, \vee)$ is a lattice with zero and unit and for any two elements x, y there is a greatest element $z = x \Rightarrow y$ such that $x \wedge z \leq y$.

It is well-known that the system $(A, 0, 1, \wedge, \vee)$ in a Heyting algebra is a distributive lattice with unit $1 = x \Rightarrow x$. The element $\neg x = x \Rightarrow 0$ is called the pseudo-complement of x .

An abstract algebra $(A, 0, 1, \wedge, \vee, \sim)$ is said to be a De Morgan algebra or quasi-Boolean algebra [1], [10, p.44], if $(A, 0, 1, \wedge, \vee)$ is a distributive lattice with zero 0 and unit 1 and \sim is an unary operation on A -called the De Morgan negation- satisfying the following conditions

$$\sim \sim x = x \quad \sim (x \vee y) = \sim x \wedge \sim y$$

We assume that the reader is familiar with the theory of Heyting and De Morgan algebras.

By a symmetrical Heyting algebra we will mean an abstract algebra $(A, 0, 1, \wedge, \vee, \Rightarrow, \sim)$ where $(A, 0, 1, \wedge, \vee, \Rightarrow)$ is a Heyting algebra and \sim is a De Morgan negation on A . We have borrowed this notion from [8]. In particular, if $(A, 0, 1, \wedge, \vee, \Rightarrow)$ is a Boolean algebra the notion of a symmetrical Heyting algebra is similar to that of a symmetrical Boolean algebra.

3. Definition and properties

Let $(A, 0, 1, \wedge, \vee, \Rightarrow, \sim)$ be a symmetrical Heyting algebra. We are going to consider $n-1$ unary operators on A (n an integer ≥ 2), noted S_1, S_2, \dots, S_{n-1} . The properties that we expect them to have are the following

- maps S_i are $(0,1)$ -lattice homomorphisms from A onto the sublattice $B(A)$ of all complemented elements of A , such that $S_i S_j x = S_j x$ for all $i, j=1, \dots, n-1$
- S_1 and S_{n-1} are respectively an interior operator and a closure operator on A [10, pp.115-116]
- they are connected with operations \Rightarrow and \sim by equations

$$S_i(x \Rightarrow y) = \bigwedge_{k=i}^{n-1} (S_k x \Rightarrow S_k y)$$

$$S_i \sim x = \sim S_{n-i} x$$

This situation can be reflected in the following definition. In the formulation below, the characterization given in [5] has played an essential rôle.

3.1 An abstract algebra $(A, 0, 1, \wedge, \vee, \Rightarrow, \sim, S_1, \dots, S_{n-1})$, n an integer ≥ 2 , where $0, 1$ are zero-argument operations $\sim, S_1, \dots, S_{n-1}$ are one-argument operations and $\wedge, \vee, \Rightarrow$ are two-argument operations is said to be a symmetrical Heyting algebra of order n if

(S1) $(A, 0, 1, \wedge, \vee, \Rightarrow, \sim)$ is a symmetrical Heyting algebra, and for every $x, y \in A$ and for all $i, j=1, \dots, n-1$ the following equations hold:

$$(S2) \quad S_i(x \wedge y) = S_i x \wedge S_i y$$

$$(S3) \quad S_i(x \Rightarrow y) = \bigwedge_{k=i}^{n-1} (S_k x \Rightarrow S_k y)$$

$$(S4) \quad S_i S_j x = S_j x$$

$$(S5) \quad S_1 x \vee x = x$$

$$(S6) \quad S_i \sim x = \sim S_{n-i} x$$

$$(S7) \quad S_1 x \vee \neg S_1 x = 1, \text{ with } \neg x = x \Rightarrow 0$$

It follows from the above definition and from the fact that the class of all symmetrical Heyting algebras is equationally definable [8] that the class of all symmetrical Heyting algebras of order n is also equationally definable.

We will refer to a SH-algebra A of order n , for short.

These structures satisfy some additional properties:

3.2 In every SH-algebra of order n the following conditions are satisfied:

$$(S8) \quad S_i 1 = 1, \quad S_i 0 = 0 \quad \text{for all } i=1, \dots, n-1$$

$$(S9) \quad S_i(x \vee y) = S_i x \vee S_i y \quad \text{for all } i=1, \dots, n-1$$

$$(S10) \quad \text{If } S_i x = S_i y \text{ for all } i=1, \dots, n-1 \text{ then } x = y \text{ (determination principle)}$$

$$(S11) \quad S_1 x \leq S_2 x \leq \dots \leq S_{n-1} x$$

The proofs of (S10) and (S11) can be found in [5].

3.3 For all $i=1, \dots, n-1$ let $S_i(A)$ be the image of A under S_i . We can show that $B(A) = S_1(A) = \dots = S_{n-1}(A)$ where $B(A)$ is the symmetrical Boolean algebra of all complemented elements.

Keeping in mind the aim mentioned in the introduction we will try to get another equivalent (but not equational) definition where the only binary operations on A will be \wedge and \vee .

3.4 A SH-algebra of order n can also be defined as a system $(A, 0, 1, \wedge, \vee, C, \sim, S_1, \dots, S_{n-1})$, n an integer ≥ 2 , where $0, 1$ are zero-argument operations, $C, \sim, S_1, \dots, S_{n-1}$ are one-argument operations and \wedge, \vee are two-argument operations such that

(T1) $(A, 0, 1, \wedge, \vee, \sim)$ is a De Morgan algebra,

and for every $x, y \in A$ and for all $i, j=1, \dots, n-1$ the following equations hold:

$$(T2) \quad S_i(x \wedge y) = S_i x \wedge S_i y$$

$$(T3) \quad S_1 x \wedge Cx = 0, \quad S_1 x \vee Cx = 1$$

$$(T4) \quad S_i S_j x = S_j x$$

$$(T5) \quad S_1 1 = 1$$

- (T6) $S_i \sim x = \sim S_{n-i} x$
 (T7) If $S_i x = S_i y$ for all $i=1, \dots, n-1$
 then $x = y$
 (T8) $S_1 x \leq S_2 x \leq \dots \leq S_{n-1} x$

Note that the formulas which make this equivalence possible are:

$$Cx = \neg S_1 x$$

$$x \Rightarrow y = y \vee \bigwedge_{j=1}^{n-1} (CS_j x \vee S_j y)$$

This expression of the \Rightarrow and properties above allow us to prove that SH-algebras of order n are in particular linearly ordered Heyting algebras, i.e. that

$$(x \Rightarrow y) \vee (y \Rightarrow x) = 1$$

holds.

4. Homomorphisms

The notion of a homomorphism from one SH-algebra of order n to a similar one, is defined as usual.

In a SH-algebra of order n every kernel of a homomorphism is a filter but, in general maximal kernels are not prime filters.

Since SH-algebras of order n are linearly ordered Heyting algebras we can point out [7] that

4.1 For every prime filter in a SH-algebra of order n , the set of prime filters containing it is linearly ordered by inclusion.

We are going to investigate some of the relationships between prime filters in a SH-algebra A of order n and ultrafilters in the symmetrical Boolean algebra $B(A)$, and that we will need in the next section.

4.2 If P is a prime filter in a SH-algebra of order n then $P' = P \cap B(A)$ is an ultrafilter in $B(A)$.

4.3 If U is an ultrafilter in the symmetrical Boolean algebra $B(A)$ then for all $i=1, \dots, n-1$ the set U_i defined by

$$a \in U_i \text{ if and only if } S_i a \in U$$

is a prime filter in A such that $U_i \cap B(A) = U$. Moreover $U_1 \subseteq U_2 \subseteq \dots \subseteq U_{n-1}$

4.4 If P is a prime filter in a SH-algebra A of order n then $P' = P \cap B(A)$ is an ultrafilter in $B(A)$ and $P'_1 \subseteq P \subseteq P'_{n-1}$.

The next statement gives a character-

ization of the prime filters in A in terms of the ultrafilters in $B(A)$.
 4.5 Theorem. If P is a prime filter in a SH-algebra A of order n then there exists a unique ultrafilter P' in $B(A)$ and an integer $i=1, \dots, n-1$ such that $P = P'_i$.

In fact, observe that by 4.2, $P' = P \cap B(A)$ is an ultrafilter in the symmetrical Boolean algebra $B(A)$ and by 4.4, $P'_1 \subseteq P \subseteq P'_{n-1}$. Let i_0 be the greatest integer $1 \leq i_0 \leq n-1$ satisfying the condition $P'_{i_0} \subseteq P$. If $i_0 = n-1$ then $P = P'_{n-1}$ and the theorem is true. If $i_0 < n-1$ we have (1) $P'_{i_0} \subseteq P$ and (2) $P'_{i_0+1} \not\subseteq P$. We will prove that $P'_{i_0} = P$. Otherwise we would have (3) $P'_{i_0} \subset P$ and by (2), 4.3 and 4.1, (4) $P \subset P'_{i_0+1}$. Let x, y be elements such that (5) $x \in P$, (6) $x \notin P'_{i_0}$, (7) $y \in P'_{i_0+1}$, (8) $y \notin P$.

Now assume $j \leq i_0$. Using (S11) and (6), $S_j x \leq S_{i_0} x$ and $S_{i_0} x \notin P'$, i.e. $S_j x \notin P'$ and by (T3), $CS_j x \in P' \subseteq P$. Consequently, $CS_j x \vee S_j y \in P$.

Suppose $i_0+1 \leq j$. By (S11) and (7), $S_{i_0+1} y \leq S_j y$ and $S_{i_0+1} y \in P'$ so $S_j y \in P' \subseteq P$ and $CS_j x \vee S_j y \in P$.

Thus for every j , $CS_j x \vee S_j y \in P$.

Hence

$$\bigwedge_{j=1}^{n-1} (CS_j x \vee S_j y) \in P \text{ and}$$

$$(9) \quad x \Rightarrow y = y \vee \bigwedge_{j=1}^{n-1} (CS_j x \vee S_j y) \in P$$

Combining (5) and (9) we get $x, x \Rightarrow y \in P$.

Since filters in Heyting algebras satisfy the Modus Ponens we infer that $y \in P$ which contradicts (8).

The uniqueness of P' follows at once from 4.3. Thus the theorem is proved.

5. n -valued SH-fields of sets

Let A be a SH-algebra of order n and E the family of all prime filters in A , ordered by inclusion.

5.1 Since A is a De Morgan algebra we can consider, following [1, p.260] the map $g: E \rightarrow E$ defined by

$$g(P) = \bigcap_A \sim P$$

for each $P \in E$, where $\sim P = \{p: \sim p \in P\}$ and C_A is the set-theoretical complement in A .

Since the operations $S_i: A \rightarrow A$, $i=1, 2, \dots, n-1$ are $(0,1)$ -lattice homomorphisms, the inverse images S_i^{-1} map prime filters in A into prime filters in A . Thus we can also define $n-1$ maps $g_i: E \rightarrow E$, $i=1, \dots, n-1$ as follows

$$g_i(P) = S_i^{-1}(P) = \{x \in A: S_i x \in P\}$$

for all $i=1, \dots, n-1$ and for each $P \in E$.

5.2 Some properties of these maps are listed below:

- (1) $g = g^{-1}$
- (2) $g_i g_j = g_i$, for $i, j=1, \dots, n-1$
- (3) $g g_i = g_{n-i} g$, for $i=1, \dots, n-1$
- (4) $E = \bigcup_{i=1}^{n-1} g_i(E)$
- (5) If $P \subseteq Q$ for $P, Q \in E$ then $g_i(P) = g_i(Q)$

It is well-known [1, p.260] that g is an involution of E , i.e. a one-one mapping from E onto E such that $g g(x) = x$, for all $x \in E$. The properties (2) and (3) are consequences of the axioms (S4) and (S6) (definition 3.2) respectively and of the properties of the inverse image. (4) is a consequence of theorem 4.5, because $P \in E$ is equivalent to the existence of an integer i such that $P = P_i' = \{x: S_i x \in P'\} = S_i^{-1}(P') = S_i^{-1}(P) = g_i(P)$. Finally let $P \subseteq Q$ so $g_i(P) \subseteq g_i(Q)$. On the other hand, let $z \in$

$g_i(Q) = S_i^{-1}(Q)$, i.e. $S_i z \in Q$, so $\neg S_i z \notin Q$ and in particular $\neg S_i x \notin P$. But $1 = S_i z \vee \neg S_i z \in P$ and P is a prime filter, hence $S_i z \in P$ and $z \in g_i(P)$.

5.3 Let T be a non-empty set and let g, g_1, \dots, g_{n-1} be n mappings from T into T fulfilling the conditions (1)-(4) of 5.2.

Let us put for each $X \subseteq T$

$$\sim X = C_T g(X)$$

$$S_i X = g_i^{-1}(X), \text{ for all } i=1, \dots, n-1$$

$$CX = C_T S_1 X$$

Let $P(T)$ be the set of all subsets of T . We are going to consider the system $(P(T), \emptyset, T, \cap, \cup, C, \sim, S_1, \dots, S_{n-1})$.

5.4 The operations defined above fulfill the properties (T1)-(T7) indicated in the definition 3.4 of a SH-algebra of order n .

(T1) has been shown in [1, p.259]. (T2) follows at once from properties of the inverse image; (T3) and (T5) follow easily. (T4) is a consequence of 5.2(2). Note that $g_i^{-1}(C_T X) = C_T g_i^{-1}(X)$ and $g(C_T X) = C_T g(X)$. By 5.2(3) we obtain (T6). Finally (T7) is a consequence of 5.2(4).

5.5 Let $Q(T)$ be a non-empty class of subsets of T containing T and closed under set-theoretical union and intersection as well as under the operations $C, \sim, S_1, \dots, S_{n-1}$ defined above. Suppose that $Q(T)$ satisfies the axiom (T8) of the definition 3.4 of a SH-algebra of order n . Thus $(Q(T), \emptyset, X, \cap, \cup, C, \sim, S_1, \dots, S_{n-1})$ is an example of a SH-algebra of order n , called an n -valued SH-field of subsets of T . We will see that n -valued SH-fields of sets are typical examples of SH-algebras of order n in the sense indicated in the next theorem.

5.6 Representation theorem. Every SH-algebra of order n is isomorphic to an n -valued SH-field of sets.

Let A be a SH-algebra of order n and let E be the set of all prime filters. For each $P \in E$ we define $g(P)$ and $g_i(P)$ as in 5.1. From 5.2 we know that the mappings g and g_i , $i=1, \dots, n-1$ satisfy the conditions (1)-(4).

For every $X \subseteq E$ let us define the operations \sim, S_i and C as in 5.3

On account of 5.4 we see that the system $(P(E), \emptyset, E, \cap, \cup, C, \sim, S_1, \dots, S_{n-1})$ fulfills the axioms (T1)-(T7) of the definition 3.4.

Following Stone [12], for every $a \in A$ we consider the map $h: A \rightarrow P(E)$ as follows: $h(a) = \{P \in E: a \in P\}$. It is well-known [1, p.260] that h is a homomorphism from the De Morgan algebra A into the De Morgan algebra $(P(E), \emptyset, E, \cap, \cup, \sim)$. In addition, for every $a \in A$, $h(S_i a) = S_i h(a)$ follows from the equivalences $P \in S_i h(a) \iff g_i(P) \in h(a) \iff a \in g_i(P) \iff S_i a \in P \iff P \in h(S_i a)$.

Now we note that for every $a \in A$,

$Ch(a) = \bigcap_{E \in S_1} h(a) = \bigcap_{E \in S_1} h(S_1 a)$. By (T3),
 $P \in h(Ca) \iff Ca \in P \iff S_1 a \in P \iff$
 $\iff P \in h(S_1 a) \iff P \in Ch(a)$.

Since h is a one-one mapping we have proved that A is isomorphic to the n -valued SH-field of sets $(\{h(a)\}, \emptyset, E, \cap, \cup, C, \sim, S_1, \dots, S_{n-1})$. The proof of the statement 5.6 is now complete.

6. Topological representation

In lattice theory, the set of all prime filters in a distributive lattice L , ordered by inclusion, does not characterize L , in spite of all the information that it gives about the structure L [3]. Here we have a similar situation. The ordered set of prime filters E with maps g, g_1, \dots, g_{n-1} as defined above, does not characterize the SH-algebra A of order n . So we need to provide E with more structure.

6.1 In his answer to the same problem for distributive lattices, Stone has endowed the set of all prime filters in a distributive lattice with a particular topology. Thus E becomes a Stone space.

In the Stone representation theory of distributive lattices there is a one-one correspondence between the family of distributive lattices and the family of Stone spaces. More precisely, each distributive lattice L is isomorphic to the family L^* of compact open sets of a Stone space $S(L)$ and the compact open sets X^* of every Stone space X form a ring of sets such that $S(X^*)$ is homeomorphic to X .

The situation for SH-algebras is described below.

6.2 We adopt the following notation. If A is a SH-algebra of order n , then E is the family of all prime filters in A and g, g_1, \dots, g_{n-1} are maps from E into E defined as in 5.1. For each $a \in A$, $a^* = \{P \in E : a \in P\}$ and $A^* = \{a^* : a \in A\}$. The maps g, g_1, \dots, g_{n-1} determine the operations \sim, C and S_i , $i=1, \dots, n-1$ in the way given in 5.3. We have shown that A and A^* are isomorphic under Stone's correspondence $x \mapsto h(x) = x^*$.

Since the maps g, g_1, \dots, g_{n-1} are defined on E we note some properties which show relations between these maps and sets of A^* .

6.3 We get

- (6) $\bigcap_{E \in S_1} g(a^*) \in A^*$, for all $a^* \in A^*$
- (7) $g_i^{-1}(a^*) \in A^*$, for all $a^* \in A^*$
- (8) $\bigcap_{E \in S_1} g_i^{-1}(a^*) \in A^*$, for all $a^* \in A^*$
- (9) $g_1^{-1}(a^*) \subseteq g_2^{-1}(a^*) \subseteq \dots \subseteq g_{n-1}^{-1}(a^*)$

We form a topological space by taking A^* as a subbase for a topology on E .

This topological space with maps g, g_1, \dots, g_{n-1} as in 5.1 will be said to be the representation space of A and will be denoted by $(S(A), g, g_1, \dots, g_{n-1})$.

A Stone space for a distributive lattice with zero and unit has been characterized (up to homomorphism) by Stone as a topological space S satisfying the following conditions [12], [9], [3]:

- (R1) S is a compact T_0 -space in which the family of compact open sets forms a base for the open sets.
- (R2) If F is a closed set in S , and $\{U_k : k \in K\}$ is a family of compact open sets of S , closed under finite intersection and such that $U_k \cap F \neq \emptyset$ for all $k \in K$, then $\bigcap (U_k : k \in K) \cap F \neq \emptyset$.
- (R3) The intersection of two compact open sets is compact.

The representation space $(S(A), g, g_1, \dots, g_{n-1})$ of a SH-algebra A of order n satisfies the following conditions:

- (1) $S(A)$ is a Stone space
- (2) The maps g, g_1, \dots, g_{n-1} from $S(A)$ into $S(A)$ fulfill the conditions (1)-(4) of 5.2
- (3) The family A^* of compact open sets of $S(A)$ satisfies the conditions (6)-(9) of 6.3

We will close our claims with a characterization of the representation space of a SH-algebra of order n . First a definition.

6.4 A SH-space of order n is a system $(X, g, g_1, \dots, g_{n-1})$ where X is a Stone space and g, g_1, \dots, g_{n-1} are n functions from X into X fulfilling the conditions (1)-(4) of 5.2 and the conditions (6)-(9) of 6.3, where A^* is the family of all compact open sets of X .

The representation space of a SH-algebra A of order n is a SH-space of order n , and A is isomorphic to the family A^* .

On the other hand, let $(X, g, g_1, \dots, g_{n-1})$ be a SH-space of order n . We can define on $P(X)$ the operations \cap, \cup and S_i , $i=1, \dots, n-1$ by the formulas of 5.3. We will show that:

6.5. The system $(A^*, \emptyset, X, \cap, \cup, \sim, S_1, \dots, S_{n-1})$ is an n -valued SH-field of sets.

In fact, by Stone's result A^* is a distributive lattice with zero and unit. Furthermore A^* is closed under the operations \cap, \cup and S_i , $i=1, \dots, n-1$, because of the definitions in 5.3 and the conditions (6)-(8) for g and g_i . The condition (9) on maps g_i means that the axiom (T8) is satisfied. The assertion is now proved.

6.6 Let $(X, g, g_1, \dots, g_{n-1})$ be a SH-space of order n and A^* the family of all compact open sets of X . Since A^* is a SH-algebra of order n we can consider the representation space $(S(A^*), g', g'_1, \dots, g'_{n-1})$ of A^* . Namely, $S(A^*)$ is the topological space consisting of the set E' of all prime filters in A^* with $(A^*)^* = \{a^* : a \in A^*\}$ as a subbase for the topology, where $a^* = \{P' \in E' : a \in P'\}$ for each $a \in A^*$. By Stone's result 6.1, the function $f(x) = P_x = \{a \in A^* : x \in a\}$ is a one-one mapping from X onto $S(A^*)$, continuous, and such that the inverse image f^{-1} (from $S(A^*)$ onto X) is also continuous. Taking in account the definitions of the functions g' and g'_i on $S(A^*)$ and the operations \sim, S_i and \cap on A^* we can show that for all $x \in X$, $fg(x) = g'f(x)$ and $fg_i(x) = g'_i f(x)$, $i=1, \dots, n-1$. So X and $S(A^*)$ are homeomorphic SH-spaces of order n . This establishes the characterization.

References

1. BIALYNICKI-BIRULA, A. and H. RASIOWA, On the representation of Quasi-Boolean algebras, Bull. Acad. Pol. Sci. 5 (1957), 259-261.
2. CIGNOLI, R., Topological representation of Łukasiewicz and Post algebras, Notas de Lógica Matemática n° 33, Universidad Nacional del Sur, Bahía Blanca, Argentina, 1974, 1-20.
3. GRATZER, G., Lattice Theory: First concepts and distributive lattices, Freeman and Co., San Francisco, 1971.
4. ITURRIOZ, L., Algèbres de Heyting trivalentes involutives, Notas de Lógica Matemática n° 31, Universidad Nacional del Sur, Bahía Blanca, Argentina, 1974.
5. ITURRIOZ, L., Łukasiewicz and symmetrical Heyting algebras, Zeitschrift für mathematische Logik und Grundlagen der Mathematik, 23 (1977), 131-136.
6. MOISIL, Gr., Logique Modale, Disquis. Math. et Phys., Bucarest, 2 (1942), 3-98.
7. MONTEIRO, A., Linéarisation de la logique positive de Hilbert-Bernays, Rev. Unión Mat. Argentina, 20 (1962), 308-309.
8. MONTEIRO, A., Sur quelques extensions du calcul propositionnel intuitionniste, IVème congrès de mathématiciens d'expression latine, Bucarest, (17-24 september 1969).
9. NERODE, A., Some Stone spaces and recursion theory, Duke Math. Jour., 26 (1959), 397-406.
10. RASIOWA, H., An algebraic approach to non-classical logics, Studies in Logic n° 78, North-Holland, Amsterdam, 1974.
11. ROUSSEAU, G., Logical systems with finitely many truth-values, Bull. Acad. Pol. Sci., 17 (1969), 189-194.
12. STONE, M., Topological representation of distributive lattices and Brouwerian logics, Casopis Pest. Mat. Fys., 67 (1937), 1-25.
13. TRACZYK, T., An equational definition of a class of Post algebras, Bull. Acad. Pol. Sci., 12 (1964), 147-149.

De Morgan Algebras - Completeness and Recursion

Louis H. Kauffman

University of Illinois at Chicago Circle

An elementary proof is given of a completeness theorem for De Morgan Algebras. The proof involves a construction that associates to a De Morgan algebra B , a new De Morgan algebra \hat{B} . The construction of \hat{B} bears a close analogy to the construction of the complex numbers from the real numbers. Similarly, De Morgan algebras may be constructed from Boolean algebras. Relationships with recursion and periodic sequences are discussed.

1. Introduction

A De Morgan algebra is an algebra that satisfies most of the properties of a Boolean algebra except for the law of the excluded middle, expressed as $x + x' = 1$ or $xx' = 0$ in the usual Boolean notation. These algebras have been studied by various authors (see [1],[2],[5]). The purpose of this paper is to give an elementary proof of a completeness theorem for De Morgan algebras, and to indicate some interesting examples of these algebras.

The completeness theorem that we prove may be deduced at once from deeper results about the structure of De Morgan algebras (see Remark 2.10). Nevertheless, I believe that the proof given here is of

interest because it is quite elementary, and it is a generalization of a corresponding argument for Boolean algebras. De Morgan algebras grow out of Boolean algebras.

In section 2 we define De Morgan Algebras and give a construction that associates to a De Morgan algebra B a new De Morgan algebra \hat{B} . The construction of \hat{B} bears a close analogy to the construction of the complex numbers from the real numbers. With the aid of this construction, the completeness result (Theorem 2.5) is proved. In section 3 we show how \hat{B} is related to recursion in B . Section 4 outlines the construction of an algebra of periodic sequences, and delineates directions for further investigation.

2. The Completeness Theorem

It is possible to choose a very concise set of axioms for the algebras we shall study. I shall give a longer list, and thereby avoid extremely detailed demonstrations.

Definition 2.1. A De Morgan algebra B is a set B together with a unary operation $a \mapsto a'$ (inversion), and two binary operations $a, b \mapsto a+b$, $a, b \mapsto ab$ that satisfy the following axioms:

- (i) The binary operations are each commutative and associative.
- (ii) $(a')' = a$, $aa = a$, $a+a = a$ for all $a \in B$.

(iii) $(a+b)' = a'b'$, $(ab)' = a'+b'$ for all $a, b \in B$.

(iv) There exist elements $0, 1 \in B$ such that $a0 = 0$, $a+0 = a$, $a1 = a$, $a+1 = 1$ for all $a \in B$.

(v) $a(b+c) = (ab)+(ac)$, $a+(bc) = (a+b)(a+c)$ for all $a, b, c \in B$.

A Boolean algebra is a De Morgan algebra that satisfies the axioms above plus

(vi) $a+a' = 1$ and $aa' = 0$ for all $a \in B$.

Definition 2.2. Let B be a De Morgan algebra and let $\hat{B} = B \times B$. Thus

$\hat{B} = \{(a,b) | a, b \in B\}$. Define operations in \hat{B} as follows:

$(a,b)(c,d) = (ac, bd)$, $(a,b)+(c,d) = (a+c, b+d)$

$(a,b)' = (b', a')$, $0 = (0, 0)$, $1 = (1, 1)$,

$\hat{i} = (1, 0)$, $\hat{j} = (0, 1)$ for $a, b, c, d \in B$.

With these definitions, it is easy to see that B becomes a De Morgan algebra. Note that $\hat{i}' = \hat{j}$ and $\hat{j}' = \hat{i}$. Hence, if B is a non-trivial ($0 \neq 1$) Boolean algebra, then \hat{B} is a De Morgan algebra that is not Boolean.

If $V = \{0, 1\}$ is the smallest non-trivial Boolean algebra, then $V = \{0, 1, \hat{i}, \hat{j}\}$ is a small De Morgan algebra. Note that $\hat{i}' = \hat{j}$, $\hat{j}' = \hat{i}$ and $\hat{i}\hat{j} = 0$, $\hat{i}+\hat{j} = 1$.

If, for $a, b, c \in B$, we adopt the convention $a(b, c) = (ab, ac)$, then we may identify $a \in B$ with $(a, a) \in \hat{B}$. In other words, the diagonal map $\Delta: B \rightarrow \hat{B}$, defined by $\Delta(a) = (a, a)$, is a De Morgan algebra homomorphism that exhibits B as a subalgebra of \hat{B} . Every element of \hat{B} is of the form $a\hat{i} + b\hat{j}$ for $a, b \in B$.

Definition 2.3. Let S be any set. The free De Morgan algebra on S , denoted $B(S)$, is obtained as follows: The primitive expressions in $B(S)$ are $0, 1$ and elements $s \in S$. Let $E(S)$ denote the set of expressions defined by the following rules:

(i) Primitive expressions are in $E(S)$. That is, $0, 1 \in E(S)$ and $S \subseteq E(S)$.

(ii) If $x, y \in E(S)$, then x' , $x+y$, and xy belong to $E(S)$.

Let \equiv denote the equivalence relation on $E(S)$ that is generated by the axioms

(i) \rightarrow (v) of Definition 2.1. Let

$B(S) = E(S)/\equiv$. It is easy to verify that $B(S)$ is a De Morgan algebra with operations inherited from the formal operations of rule (ii).

Definition 2.4. A homomorphism $\phi: B \rightarrow C$ of De Morgan algebras B and C is a set-mapping such that $\phi(xy) = \phi(x)\phi(y)$, $\phi(x+y) = \phi(x) + \phi(y)$ and $\phi(x') = \phi(x)'$ for all $x, y \in B$.

As usual, a homomorphism from a free algebra $B(S)$, $\phi: B(S) \rightarrow C$, is determined uniquely by its values $\phi(s)$ for $s \in S$.

We are now prepared to state the completeness theorem.

Theorem 2.5. Let $\hat{V} = \{0, 1, \hat{i}, \hat{j}\}$ be the four element De Morgan algebra described after Definition 2.2. Let $B(S)$ be a free De Morgan algebra on a set S . Then for $\alpha, \beta \in E(S)$, $\alpha = \beta$ if and only if $\phi(\alpha) = \phi(\beta)$ for every homomorphism $\phi: B(S) \rightarrow \hat{V}$.

In other words, an equality $\alpha = \beta$ is a consequence of the axioms for a De Morgan algebra if and only if it is true about the model \hat{V} .

In order to prove Theorem 2.5 we shall need some preliminary lemmas.

Lemma 2.6. Let $f(x) \in E(S)$ be an expression involving $x \in S$. Then $f(x) = Ax + Bx' + Cxx' + D$ where A, B, C, D are expressions that do not contain the element x .

The proof of this lemma proceeds just as in ordinary Boolean algebra, and will therefore be omitted.

Lemma 2.7. The following equivalence holds in $E(S)$:

$$(Ax + Bx' + Cxx')' = A'x + B'x' + xx' + A'B'C'$$

Proof: $(Ax + Bx' + Cxx')'$

$$\begin{aligned} &= (Ax)'(Bx')'(Cxx')' \\ &= (A' + x')(B' + x'')(C' + (xx')') \\ &= (A' + x')(B' + x)(C' + x' + x) \\ &= (A'B' + A'x + B'x' + x'x)(C' + x' + x) \\ &= A'B'C' + (A'C' + A'B' + A')x \\ &\quad + (B'C' + A'B' + B')x' \\ &\quad + (A' + B' + C' + 1 + 1)xx' \\ &= A'B'C' + A'(C' + B' + 1)x \\ &\quad + B'(C' + A' + 1)x' + xx' \\ &= A'B'C' + A'x + B'x' + xx' \end{aligned}$$

This completes the proof of the lemma.

For the next lemma, we shall proceed informally, regarding $f(x)$ as a function of x , and evaluating $f(0)$, $f(1)$, $f(i)$, $f(j)$. This can all be made more precise in terms of homomorphisms, but only at some loss in clarity.

Lemma 2.8. Let $f(x) = Ax + Bx' + Cxx' + D$ be an element of $E(S)$ as described in Lemma 2.6. Then

$$\begin{aligned} f(0) &= B + D \\ f(1) &= A + D \\ (f(i) + i)(f(j) + j) &= D \\ f(i) + f(j) &= A + B + C + D. \end{aligned}$$

Here equality means as functions on \hat{V} .

Proof: Certainly $f(0) = B + D$ and $f(1) = A + D$. Now

$$\begin{aligned} f(i) &= (A + B + C)i + D \text{ (since } i' = \bar{i} \text{)}, \text{ and} \\ f(j) &= (A + B + C)j + D. \text{ Hence} \\ f(i) + i &= (A + B + C + 1)i + D = i + D, \end{aligned}$$

and $f(j) + j = j + D$. Therefore

$$\begin{aligned} (f(i) + i)(f(j) + j) &= (i + D)(j + D) \\ &= ij + iD + jD + D \\ &= 0 + (i + j)D + D \\ &= D + D = D. \end{aligned}$$

$$\begin{aligned} \text{Finally, } f(i) + f(j) &= (A + B + C)(i + j) + D \\ &= A + B + C + D. \end{aligned}$$

This completes the proof of the lemma.

Proof of Theorem 2.5. Let $\alpha, \beta \in E(S)$.

By Lemma 2.6 we may assume that $\alpha = Ax + Bx' + Cxx' + D$ and that $\beta = \bar{A}x + \bar{B}x' + \bar{C}xx' + \bar{D}$ where A, B, C, D and $\bar{A}, \bar{B}, \bar{C}, \bar{D}$ are expressions that do not contain x . The proof will proceed by induction on the total number N of variables (elements of S) that occur in the two expressions. If $N=0$, then $\alpha = D$ and $\beta = \bar{D}$ where D and \bar{D} are equal to 0 or 1. Since $0 \neq 1$ in a free De Morgan algebra, the theorem is trivial for the case $N=0$. If $N > 0$, then either α or β contains a variable x and we may use the equivalence indicated above.

Thus we may assume by induction (using Lemma 2.8) that the following equivalences hold in $E(S)$:

$$(*) \quad \begin{cases} B + D = \bar{B} + \bar{D} \\ A + D = \bar{A} + \bar{D} \\ D = \bar{D} \\ A + B + C + D = \bar{A} + \bar{B} + \bar{C} + \bar{D} \end{cases}$$

We now use $(*)$ to show that $\alpha = \beta$ in $E(S)$.

$$\begin{aligned} \alpha &= Ax + Bx' + Cxx' + D \\ &= (A'x + B'x' + xx' + A'B'C')' + D, \\ \text{by (2.7)} \\ &= (A'x)'(B'x')'(xx')'(A'B'C')' + D \\ &= ((A'x)' + D)((B'x')' + D)((xx')' + D) \\ &\quad ((A'B'C')' + D) \\ &= ((A + D) + x)((B + D) + x)((xx')' + D) \\ &\quad + D)(A + B + C + D) \end{aligned}$$

Now make the substitutions indicated by $(*)$, reverse steps, and conclude that $\alpha = \beta$. This completes the induction step and the proof of the theorem.

Note that Theorem 2.5 has the following corollary:

Corollary 2.9. Let $B(S)$ be a free De Morgan algebra. Then $B(S)$ is isomorphic to a sub-algebra of a product of copies of the four element algebra $\hat{V} = \{0, 1, i, j\}$.

Proof: Let $\Phi = \{\phi: B(S) \rightarrow \hat{V} \mid \phi \text{ is a homomorphism}\}$. Let \hat{V}_ϕ be a copy of \hat{V}

indexed by an element of \mathbb{Z} , and let $v = \prod_{\phi \in \mathbb{Z}} \hat{V}_\phi$. Define $F : B(S) \rightarrow v$ by

$$F(\alpha) = \prod_{\phi \in \mathbb{Z}} \phi(\alpha) \text{ where } \phi : B(S) \rightarrow \hat{V}_\phi. \text{ Then}$$

by 2.5 $F(\alpha) = F(\beta)$ if and only if $\alpha = \beta$. Hence F injects $B(S)$ as a subalgebra of v .

Remark 2.10. In fact, Corollary 2.9 is true for arbitrary De Morgan algebras. This deeper result may be found in [2] or [5].

3. Recursion and Fixed Points

The \wedge construction leading from a Boolean algebra B to its corresponding De Morgan algebra \hat{B} is closely related to the structure of recursion in the given Boolean algebra.

Let $T : B \rightarrow B$ be a mapping of the form $T(x) = ax + bx'$ where $a, b \in B$. T has period two; in fact

$$T^2(x) = T(T(x)) = (a + b)x + abx'$$

$$T^3(x) = ax + bx' = T(x)$$

...

$$T^{n+2}(x) = T^n(x).$$

Hence there may be no element $X \in B$ such that $T(X) = X$. However, \hat{B} does contain such fixed points. For example, if $T(x) = x'$, then $x = x'$ has no solutions in the Boolean algebra B , but is satisfied by \hat{i} and \hat{j} in \hat{B} .

Proposition 3.1. Let B be a Boolean algebra, and let $T : B \rightarrow B$ be the mapping described above. Let $\mathcal{T} : \hat{B} \rightarrow \hat{B}$ be the corresponding mapping on \hat{B} . Then there exist elements X of \hat{B} such that $\mathcal{T}(X) = X$. In particular, we may take $X = (T(x), T^2(x))$ or $X = (T^2(x), T(x))$ for any $x \in B$.

Proof: The following identities in B are easily verified:

$$T(x) = aT(x) + bT^2(x)', \quad T^2(x) = aT^2(x) + bT(x)'$$

Let $X = (T(x), T^2(x))$. Then

$$\begin{aligned} \mathcal{T}(X) &= aX + bX' \\ &= a(T(x), T^2(x)) + b(T^2(x)', T(x)') \\ &= (aT(x) + bT^2(x)', aT^2(x) + bT(x)') \\ &= (T(x), T^2(x)) \\ &= X. \end{aligned}$$

Thus the algebraic structure of \hat{B} reflects the properties of period two sequences recursively generated from B . The next section describes a more general De Morgan algebra of sequences.

4. Sequence Models

Corresponding to a De Morgan algebra B let $\mathcal{S}(B)$ denote the set of all sequences of elements of B with an assigned even period (possibly of period 0). That is $\mathcal{S}(B)$ consists of sequences $b = \{b_n\}$ such that n ranges over the integers and $b_{n+p} = b_n$ for all n , where $p = p(b)$ is an even non-negative integer associated with the sequence.

$\mathcal{S}(B)$ has the structure of an algebra as follows:

- (i) $(ab)_n = (a_n b_n)$, $(a + b)_n = a_n + b_n$
and $p(ab) = p(a + b) = \text{lcm}(p(a), p(b))$
if $p(a) \neq 0$
and $p(b) \neq 0$,
0 otherwise,

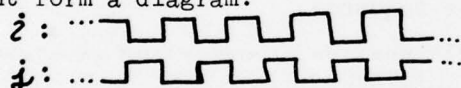
Here the symbol lcm denotes least common multiple.

- (ii) $(a')_n = (a_{n-k})'$ where $k = p(a)/2$,
 $p(a') = p(a)$.

Note that in the sequence algebra, inversion is obtained by ordinary inversion plus a half-period shift. The sub-algebra of period two sequences in $\mathcal{S}(B)$ is isomorphic to B . As it stands, the sequence algebra is not a De Morgan algebra.

Axioms (i), (ii), (iii), and (v) are satisfied but there is no available choice for 0 or 1. If $\mathfrak{g}(B)_p$ denotes the subalgebra of sequences of period p then we may take 0_p and 1_p to be the constant sequences of zeroes and ones respectively, with assigned period p . This gives $\mathfrak{g}(B)_p$ the structure of a De Morgan algebra for each p . We could force $\mathfrak{g}(B)$ into the mold by taking a quotient construction, but I believe it is more interesting to leave it as it stands. Our rules for combining sequences of different periods provide a simple model of interference phenomena that bears investigation on its own grounds.

We may regard $\mathfrak{g}(B)$ as a set of periodic oscillations. In this regard it is interesting to compare our ideas with the suggestions of G. Spencer Brown in his book Laws of Form ([4]). Spencer Brown suggests that an element X satisfying $X' = X$ might be seen as an oscillation (if its 0, then its 1, then its 0, then its 1, ...). Taking this literally, we might form a diagram:



In each case, the spatial sense in which $\dot{z}' = \dot{z}$ and $\dot{j}' = \dot{j}$ involves ordinary inversion plus a left-right shift. With \dot{z} and \dot{j} synchronized (and note that here the synchronization has become the spatial relationship of the two sequences) as above, we have $\dot{z}\dot{j} = 0$. This is the motivation for our construction of $\mathfrak{g}(B)$, and also for the construction $B \mapsto \hat{B}$ of section 2.

There are many connections between our discussion and Brown's work. These will be explored in another paper. I would like to remark here that it is striking that once one lifts the law of the excluded middle from Boolean algebra, there is opened up the possibility of infinite models involving simple analogs of wave-

forms and interference phenomena. This temporal, musical aspect is precisely what is prohibited by the stark all or nothing of two-valued logic. When we drop these restrictions, the result is not fuzziness and ambiguity, but the precise emergence of patterned forms, spatial and temporal.

Bibliography

- [1] Bialynicki-Birula, A. and Rasiowa, H., On the representation of quasi-Boolean algebras. Bull. Acad. Polon. Sci. Ser. Sci. Math. Astronom. Phys. 5 (1957), 259-261.
- [2] Balbes, R. and Dwinger, P., Distributive Lattices, Univ. of Missouri Press (1974).
- [3] Berman, J. and Dwinger, P., De Morgan algebras - free products and free algebras, (unpublished).
- [4] Brown, G. Spencer, Laws of Form, The Julian Press, Inc., New York (1972).
- [5] Kalman, J. A., Lattices with involution, Trans. Amer. Math. Soc. 87 (1958), 485-491.
- [6] Kauffman, L. H., Network synthesis and Varela's calculus, (to appear in Int. J. Gen. Syst.).
- [7] Varela, F. J., A calculus for self-reference, Int. J. Gen. Syst. 2 (1975), 5-24.
- [8] Varela, F. J., The arithmetic of closure, Progress in Cybernetics and Systems Research, Vol. 3, edit. by R. Treppel, G. Klir and L. Ricciardi, Hemisphere Publications (Wiley) N.Y. (1977).

ON THE COMPACTIFICATION AND ENUMERATION OF DISTINCT FUZZY SWITCHING FUNCTIONS

Abraham Kandel
Computer Science Department

New Mexico Institute of Mining and Technology
Socorro, New Mexico 87801

ABSTRACT

In this paper we are concerned with the study of compactification and enumeration of fuzzy switching functions. We use compactness in the sense of achieving a greater degree of simplification in the mode of representation of these functions, similarly to the way used in Zadeh (1976).

Novel techniques for the enumeration of fuzzy switching functions are discussed. The approach described in this paper provides a framework for solving the enumeration problem through the use of combinatorial arguments.

2. ELEMENTARY PROPERTIES OF FUZZY ALGEBRA

Let $X = \{x\}$ denote a space of objects. Then a fuzzy set A in X is a set of ordered pairs $A = \{(x, \mu_A(x))\}, x \in X$ where $\mu_A(x)$ is termed the grade membership of x in A . We shall assume for simplicity that $\mu_A(x)$ is a number in the interval $[0, 1]$, with the grades 1 and 0 representing respectively, full membership and nonmembership in a fuzzy set.

Definition 1: A fuzzy algebra is the system $F = \langle Z, +, *, -, \bar{} \rangle$ where F has at least two distinct elements, and $\forall x, y, z \in Z$, system F satisfies the following set of axioms:

- (1) Idempotency: $x+x=x$ $x*x=x$
- (2) Commutativity: $x+y=y+x$ $x*y=y*x$
- (3) Associativity: $(x+y)+z=x+(y+z)$
 $(x*y)*z=x*(y*z)$
- (4) Absorption: $x+(x*y)=x$ $x*(x+y)=x$
- (5) Distributivity: $x+(y*z)=(x+y)*(x+z)$
 $x*(y+z)=x*y+x*z$
- (6) Complement: If $x \in Z$ then there is a unique complement \bar{x} of x such that
 $x \in Z$ and $\bar{\bar{x}}=x$.
- (7) Identities: $(\exists! e_+)(\forall x)$ such that $x+e_+=e_++x=x$
 $(\exists! e_*)(\forall x)$ such that $x*e_*=e_**x=x$
- (8) De-Morgan Laws: $\overline{x+y}=\bar{x}*\bar{y}$ $\overline{x*y}=\bar{x}+\bar{y}$

The system is a distributive lattice with existence of unique identities under $+$ and $*$. It is noted that a Boolean algebra is a complemented distributive lattice with existence of unique identities under $+$ and $*$. However, for every element x in Boolean algebra there exists a unique complement, \bar{x} , such that $x\bar{x} = 0$ and

$x+\bar{x}=1$, which is not so in fuzzy algebra. Hence, every Boolean algebra is a fuzzy algebra, but not vice versa.

In this case we will use a particular fuzzy algebra defined by the system

$$F = \langle [0, 1], +, *, -, \bar{} \rangle,$$

where $+$, $*$, and $-$ are interpreted as Max, min and complement ($\bar{x} = 1-x$, $\forall x, x \in [0, 1]$), respectively. The unique identities e_+ and e_* are 0 and 1, respectively.

Let A and B be two fuzzy sets in X . The following terminology parallels closely the presentation of fuzzy sets by Zadeh [1965].

Equality ($A=B$) is defined by

$$A = B \leftrightarrow \mu_A(x) = \mu_B(x), \forall x \in X.$$

Fuzzy set A is **contained** in B ($A \subset B$) if

$$\mu_A(x) \leq \mu_B(x), \forall x \in X. \text{ A fuzzy set } \bar{A} \text{ is the } \underline{\text{complement}}$$

of a fuzzy set A if $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$, $\forall x \in X$.

The **union** of two fuzzy sets A and B in X is defined as the membership function of $A+B$ given by

$$\mu_{A+B}(x) = \text{Max}[\mu_A(x), \mu_B(x)].$$

The **intersection** of A and B in X , denoted by $A*B$ is defined similarly by

$$\mu_{A*B}(x) = \text{min}[\mu_A(x), \mu_B(x)].$$

In the sequel, the term "fuzzy variable" will replace the term membership grade of a variable in a set. Conventionally we shall drop the $*$ symbol, i.e., $x*y$ will be written as xy .

We can now define fuzzy forms, generated by x_1, \dots, x_n , recursively as follows:

- a) The numbers 0 and 1 are fuzzy forms.
- b) A fuzzy variable x_i is a fuzzy form.
- c) If A is a fuzzy form, then \bar{A} is a fuzzy form.
- d) If A and B are fuzzy forms, then $A+B$ and AB are fuzzy forms.
- e) The only fuzzy forms are those given by rules (a) through (d).

As in two valued logic, we shall merge the concepts of fuzzy functions and fuzzy forms, to an extent, by representing the mapping by fuzzy forms.

3. FUZZY LATTICES

A lattice L is defined to be a partially ordered set with the property that every two elements of the poset have a greatest lower bound (glb) and a least upper bound (lub). (The glb and the lub of any two elements may be found by taking the intersection and the union, respectively, of the two elements.) The poset is usually represented diagrammatically for simple posets by placing the elements of the set at the vertices of a graph; lines between the vertices connect two elements to their glb and their lub.

An atom of a lattice is any element of that lattice which "covers" (\succ) the minimal element(s) of the lattice -- i.e., there is no element between the minimal element and any atom. More precisely, let a be an element of L . Then a is an atom iff $a \neq 0$ and for all elements x of L , either $ax = 0$ or $ax = a$. In other words, the minimum of an atom and any other element of the lattice must be either that atom or else zero.

The lattice representing a fuzzy function of n variables may be constructed in a manner similar to that used in constructing a Boolean lattice, although the process is much more tedious because the number of vertices increase at a much greater rate.

From combinatoric arguments it has been shown [Kandel, 1974], that there are 2^{4^n} possible fuzzy functions of n variables. However, most of these are simply non-minimized forms of a relatively few number of unique fuzzy functions. The problem can be appreciated best by considering the number of functions of two variables that exist: for Boolean

functions there are 2^{2^2} , or 16; for fuzzy functions though, there are 2^{4^2} , or 65,536. It is obviously impossible to enumerate them by hand, which in the case of Boolean functions is quite reasonable for small n . We will return to the problem of the number of fuzzy functions shortly.

Proposition: The unique atom of the lattice representing all fuzzy functions of n variables is the minterm $x_1 \bar{x}_1 x_2 \bar{x}_2 \dots x_n \bar{x}_n$.

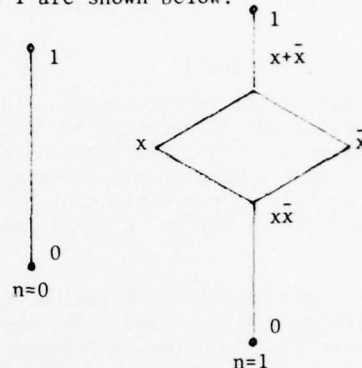
Proof: We will first show that the minterm $x_1 \bar{x}_1 x_2 \bar{x}_2 \dots x_n \bar{x}_n$ is indeed an atom. By the definition of the atom, for all other elements a of the fuzzy lattice, $ax_1 \bar{x}_1 \dots x_n \bar{x}_n$ must be either 0 or the atom itself. Obviously if $a=0$, then the product will be 0; if a is non-zero, then the product must be the term $x_1 \bar{x}_1 \dots x_n \bar{x}_n$ again by the rules of minimum and maximum (alternatively, $x_1 \bar{x}_1 \dots x_n \bar{x}_n$ must contain all the factors of a ; therefore, their product will be the atom itself).

To show the uniqueness of this atom, consider the following consequence of the basic definition of an atom of a lattice: if a and b are atoms, then if $ab \neq 0$, $a = b$. We have shown that there exists at least one atom; let it be a . The above

proof demonstrates that for $b \neq 0$, $ab = a$. Since $a \neq 0$, $ab \neq 0$, so $a = b$. Therefore, any other atom possible turns out to be the same atom.

From the uniqueness of this atom we can conclude that the diagram of the lattice for fuzzy functions of n variables has the above fundamental phrase directly above (distance 1 from) the "bottom" of the graph (zero).

The lattices for fuzzy functions with $n = 0$ and $n = 1$ are shown below.



The size of the lattice - the number of vertices which it has - is a direct measurement of the number of fuzzy functions of n variables that exist. Before being able to specify the lattice for a generalized case (instead of the specific cases given above), it will be necessary to determine exactly how many unique fuzzy switching functions there are.

4. MINIMIZATION AND ENUMERATION

Theorem 1: Let P be a phrase of fuzzy literals from the set $\{x_i\}_{i=1}^n$. A disjunction \hat{a} of any variable x_k and its complements \bar{x}_k , $1 \leq k \leq n$, can be appended to P without affecting the general value of the phrase iff there exists a variable x_i and its complement \bar{x}_i in P , for some i , $1 \leq i \leq n$.

Proof: (a) Assume x_i and \bar{x}_i in P . ($P = \alpha x_i \bar{x}_i \gamma$, where α , β , and γ are conjunctions of literals from the set $\{x_i\}_{i=1}^n$.) Obviously, $x_i \bar{x}_i \leq 0.5$ and thus $P \leq 0.5$. However, $\alpha \geq 0.5$ and therefore the "if" part is proved.

(b) Assume that $x_i \bar{x}_i$ for some i , $1 \leq i \leq n$ is not in P . Then assign grades of membership larger than 0.5 to all literals of P . Clearly $P > 0.5$. Q.E.D.

Similarly, we can prove the dual theorem.

Theorem 2: Let C be a clause of fuzzy literals from the set $\{x_i\}_{i=1}^n$. A conjunction \hat{b} of any variable x_k and its complement \bar{x}_k , $1 \leq k \leq n$, can be appended to C without affecting the general value of the clause iff there exists a variable x_i and

its complement \bar{x}_i in C, for some i , $1 \leq i \leq n$.

In general if F is a conjunction of formulas it can take a value < 0.5 if certain conditions are satisfied. Clearly, if F is of the form

$F = [\sum_j x_j \bar{x}_j \gamma_j] \beta$, $1 \leq j \leq n$ when β and $\{\gamma_j\}_{j=1}^n$ are formulas in $\{x_i\}_{i=1}^n$ then $F \leq 0.5$. This is a trivial case where one formula in the conjunction is < 0.5 and thus F is < 0.5 . A more general case can be proven as follows.

Theorem 3: Let the set $\{F_i\}_{i=1}^{\omega}$ be a set of fuzzy formulas over x_1, x_2, \dots, x_n , and let F be a conjunction of formulas from this set. A disjunction F_d of any formula F_k and its complement \bar{F}_k , can be appended to or deleted from the conjunction representing F, without affecting the value of F, if there exists functions F_s and F_t in the conjunction representing F such that \bar{F}_s is subsumed by F_t .

Proof: Let F_s and F_t be in the conjunction representing F such that F_s is subsumed by F_t . Then $F_s F_t \leq 0.5$ since

$$F_s \bar{F}_t = \bar{F}_s + \bar{F}_t = \bar{F}_s + F_t + \bar{F}_t \geq 0.5. \text{ Q.E.D.}$$

Theorem 4: Let the set $\{F_j\}_{j=1}^t$ be a set of fuzzy formulas over x_1, x_2, \dots, x_n , and let F be a disjunction of formulas from this set.

A conjunction F_c of any formula F_r and its complement \bar{F}_r , can be appended to or deleted from the disjunction representing F, without affecting the value of F, if there exist functions F_i and F_1 in the disjunction representing F such that F_i is subsumed by \bar{F}_1 .

Proof: $F_i + F_1 \geq 0.5$ since $F_i + F_1 = F_i + \bar{F}_1 + F_1$.
Q.E.D.

It should be noted that these formulas can be generated by combining several subformulas under the rules of fuzzy algebra. Thus the sets

$\{F_j\}_{j=1}^{\omega, t}$ are compositions or decompositions of a variety of fuzzy forms.

Example 1: $f(x_1, x_2) = x_1 x_2 + x_1 \bar{x}_2$. The terms $x_1 x_2$ and $x_1 \bar{x}_2$ are obviously fuzzy prime implicants of $f(x_1, x_2)$. The consensus as originally defined will produce $x_1 x_2 \psi x_1 \bar{x}_2 = 0$. However, the term $x_1 \bar{x}_1$ can be added to $f(x_1, x_2)$ without changing the function and it subsumes no other fuzzy implicant of $f(x_1, x_2)$. Hence

$$f(x_1, x_2) = x_1 x_2 + x_1 \bar{x}_2 + x_1 \bar{x}_1$$

where this is the fuzzy prime implicant representative of the function. Clearly,

$$x_1 \bar{x}_1 = x_1 \bar{x}_1 x_2 + x_1 \bar{x}_1 \bar{x}_2$$

and thus

$$f(x_1, x_2) = x_1 x_2 + x_1 \bar{x}_2 + x_1 \bar{x}_1 x_2 + x_1 \bar{x}_1 \bar{x}_2$$

and $f_{\min}(x_1, x_2) = x_1 x_2 + x_1 \bar{x}_2$. Namely, $x_1 \bar{x}_1$ is not an essential fuzzy prime implicant.

Definition 2: Let R and Q be two phrases over the set of fuzzy variables x_1, x_2, \dots, x_n . The fuzzy

consensus of R and Q, written $R \psi Q$, is defined to be the set of phrases $\{R_i Q_i\}$, where $R = x_i R_i$ and

$Q = \bar{x}_i Q_i$ (or $R = \bar{x}_i R_i$ and $Q = x_i Q_i$) and

$x_i \in \{x_1, x_2, \dots, x_n\}$, if the phrase $R_i Q_i$ include the conjunction $x_j \bar{x}_j$ for at least one j , $j \in \{1, 2, \dots, n\}$.

If the phrase $R_i Q_i$ does not include $x_j \bar{x}_j$ for any j , $j \in \{1, 2, \dots, n\}$, then

$$R \psi Q = \{R_i Q_i x_j \bar{x}_j | j=1, 2, \dots, n, x_i \in \{x_1, x_2, \dots, x_n\}\}.$$

If none of the above occurs then we say that $R \psi Q = 0$. Clearly if $R = x_i R_i$ and $Q = \bar{x}_i Q_i$ (or $R = \bar{x}_i R_i$ and $Q = x_i Q_i$)

and $R_i Q_i$ does not include $x_j \bar{x}_j$ for any j ,

$j \in \{1, 2, \dots, n\}$. Then $R + Q + (R \psi Q) = R + Q$.

We shall define two kinds of fuzzy phrases.

The first kind, to which we shall refer as type-1 phrase, are phrases which contain a conjunction of the form $x_j \bar{x}_j$ for at least one j , $j \in \{1, 2, \dots, n\}$.

Otherwise we refer to the phrase as a type-2 phrase. Clearly a type-1 phrase cannot be subsumed by type-2 phrases. However, they can subsume some of them. For the case where members of the set $\{R_i Q_i\}$ do not

include a conjunction of the form $x_j \bar{x}_j$ for at least one j , $j \in \{1, 2, \dots, n\}$, two situations must be checked:

(a) R and Q are both type-2 phrases. Since $\{R_i Q_i x_j \bar{x}_j | j=1, 2, \dots, n\}$ is a set of type-1 phrases covered by $R + Q$, this set is not needed.

(b) R is a type-1 phrase and Q is a type-2 phrase. In order for members of $\{R_i Q_i\}$ not to include a

conjunction $x_j \bar{x}_j$ for any j , $j \in \{1, 2, \dots, n\}$, R must be of the form $\alpha x_i \bar{x}_i \beta$ and Q must be of the form

$\gamma x_i \delta$ (or $\gamma' \bar{x}_i \delta'$) where the phrase $\alpha x_i \beta \gamma \delta$

(or $\alpha \bar{x}_i \beta \gamma' \delta'$) is a type-2 phrase. Thus

$$R_i = \alpha x_i \beta \quad (\text{or } \alpha \bar{x}_i \beta),$$

$$Q_i = \gamma \delta \quad (\text{or } \gamma' \delta').$$

and obviously the set $\{R_i Q_i x_j \bar{x}_j | j=1, 2, \dots, n\}$ is covered by Q and is not needed.

We define a fundamental phrase as either a phrase of type-2 or phrases of type-1 containing at least one literal of each variable. Clearly each phrase of type-2 in a fuzzy switching function is an essential prime implicant [see Kandel (1973), and Kandel (1974)].

Kameda and Sadeh [1977] have shown that there are $\binom{n}{k}$ ways of selecting k out of n variables and each variable may be either complemented or left uncomplemented. Thus the number of phrases type-2 containing k literals is $\binom{n}{k} 2^k$. It follows that the total number of phrases type-2 is

$$\sum_{k=1}^n \binom{n}{k} 2^k = (1+2)^n - 1 = 3^n - 1.$$

As for the fundamental phrases of type-1, each such phrase contains, for each k , either x_k only, \bar{x}_k only, or $x_k \bar{x}_k$. Thus there are 3^n such phrases. Since 2^n of them are simple, altogether there are $2 \cdot 3^n - 2^n - 1$ fundamental phrases. Clearly, any function can be defined by disjunction of a non-empty subset of the set of all fundamental phrases. therefore there are at most $2^{2 \cdot 3^n - 2^n - 1} - 1$ fuzzy switching functions.

Following Kamdea and Sadeh (1977) we call a set of phrases "independent" iff none of the phrases in the set implies any other in the same set.

We use this definition in order to generate an independence relation R^* between the fundamental phrases. This relation can be graphically represented as an upper diagonal binary matrix showing the independence relation of the $2 \cdot 3^n - 2^n - 1$ elements.

Thus the number of distinct fuzzy switching functions will be given by M_n , when

$$M_n = 2 \cdot 3^n - 2^n - 1 + \sum_{i=1}^{\text{MAX}} t \cdot \ell_i$$

and $t \cdot \ell_i$ is transitivity level i on the matrix, and MAX will denote the maximum level of transitivity for this particular n , given by the maximum cover fraction discussed by Schwede [1976] and Schwede and Kandel [1977].

Converting M_n to an explicit function of n will not be discussed here. However, it should be noted that for $n=2$, we get directly that $M_2=82$ where the Kameda and Sadeh [1977] technique yields an upper bound of 8192 and a lower bound of 49.

It should be noted that the problems in fuzzy switching algebra have many features in common with the construction of branching questionnaires, feature selection in pattern recognition, and the optimization of decision tables; (for details, see Zadeh [1976]).

It is only our hope that the ideas and the results presented in this paper are merely a step in the application of fuzzy switching structures to related fields as discussed above.

REFERENCES

- Kameda, T. and E. Sadeh, (1977), Bounds on the number of fuzzy functions, Information and Control 35, pp. 139-145.
- Kandel, A. (1973), On minimization of fuzzy functions, Trans. IEEE Computers C-22, 826-832.
- Kandel, A. (1974), On the properties of fuzzy switching functions, J. Cybernetics 4(1), 119-126.
- Schwede, G.W. (1976), N-variable fuzzy maps with application to disjunctive decomposition of fuzzy switching functions, Proceedings of the Sixth Int.

Symp. on MVL, 203-216.

Schwede, G.W., and A. Kandel (1977), Fuzzy maps, IEEE Trans. SMC 7, 669-674.

Zadeh, L.A. (1965), Fuzzy sets, Information and Control 8, 338-353.

Zadeh, L.A. (1976), A fuzzy - algorithmic approach to the definition of complex or imprecise concepts, Inf. J. Man-Machine Studies 8, 248-291.

AD-A055 763

ILLINOIS INST OF TECH CHICAGO DEPT OF COMPUTER SCIENCE F/G 12/1
PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED L--ETC(U)
MAY 78 A S WOJCIK, R SWARTWOUT N00014-78-G-0019

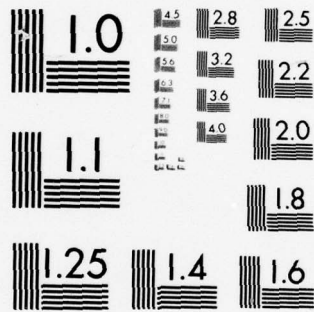
UNCLASSIFIED

NL

2 OF 4

AD
A055763





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

FUZZY FOUR-VALUED LOGIC FOR INCONSISTENCY AND UNCERTAINTY

Mark E. Stickel

Department of Computer Science
The University of Arizona
Tucson, Arizona 85721

Abstract

A logic with truth values represented by points in the unit square is developed. The logic is simultaneously a strict generalization of Belnap's four-valued logic for handling inconsistency and of Zadeh's fuzzy logic for handling uncertainty. The logic can be used for a question answering system or data base of fuzzy information in which inconsistent data may appear but should not result in everything being reported as true.

1. Introduction

A classic problem with the use of propositional or predicate calculus in the construction of question answering systems or data bases is the fact that any inconsistency, no matter how apparently irrelevant, logically implies everything.

A solution to this problem for ordinary truth values T and F was developed by Belnap^{1,2}. This solution involves the extension of conventional two-valued logic to a four-valued logic with truth values T, F, Both, and None.

This four-valued extension of conventional two-valued logic is extended in this paper to fuzzy logic where the truth values T and F are fuzzified to be real numbers in the range [0,1]. Truth values for four-valued fuzzy logic are of the form $\langle x, y \rangle$ where each of x and y is in the range [0,1], x denotes degree of asserted truth of the assertion, and y denotes degree of asserted falsity of the assertion.

The following section briefly describes Belnap's four-valued logic in preparation for the definition of four-valued fuzzy logic. The form of the description differs somewhat from that of Belnap though the semantics do not.

2. Belnap's four-valued logic

Belnap^{1,2} developed a four-valued logic for dealing with inconsistency. His model is a question answering system (QAS) which accepts inputs and answers queries. However, user inputs are not regarded as being totally reliable, and it is assumed there is no way to distinguish correct inputs from incorrect inputs by, for example, time order or source. Since the inputs are generally reliable, they are incorporated into the QAS's data base uncritically. Queries relating to inconsistent data will result in responses noting the inconsistency; unrelated queries will be answered on the basis of relevant information in the data base, not merely answered affirmatively as in classical logic in which everything can be proved from an inconsistency.

Belnap's method of handling inconsistency involves extending the classical two-valued logic to a four-valued logic. The four values are T (indicating the system has been told the proposition is true), F (indicating the system has been told the proposition is false), Both (indicating the system has been told both the proposition is true and it is false), and None (indicating the system has been told none of the proposition is). The approximation lattice A4 in Figure 1 represents relationships among the four truth values. Here, Both represents maximum (totally inconsistent) information about a proposition, None represents minimum (no) information, and T and F represent incomparable intermediate amounts of information.

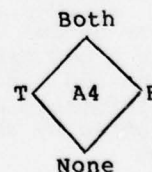


Figure 1

When only atomic formulas are input to the QAS, its knowledge state can be represented by a single set-up: a list of atomic formulas and their values. When an atomic formula is input with any of the four values (ordinarily just T or F) its value in the new knowledge state is the join (\cup_A) in the A4 lattice of the new value and its previous value.

This is expressed by the truth table in Figure 2.

		Input Value			
		Both	T	F	None
O l d V a l u e	Both	Both	Both	Both	Both
	T	Both	T	Both	T
	F	Both	Both	F	F
	None	Both	T	F	None

Figure 2

When the QAS is queried for the value of an atomic formula, its value is merely looked up in the set-up. Answers to compound queries are computed from values of their atomic components using the truth tables in Figure 3.

The latter two truth tables expressing the computation of values for the \vee and \wedge connectives can also be expressed by the logical lattice L4 in Figure 4 where the join operation \cup_L computes the \vee connective and the meet operation \cap_L computes the \wedge connective.

Let a set-up s be represented as a set of ordered pairs with distinct first elements (p, v) of atomic formula p with value v . Then the function s which computes truth values assigned by set-up s is defined recursively by

$$\begin{aligned} s(A) &= \text{the } v \text{ such that } (A, v) \in s \\ &\quad \text{when } A \text{ is atomic} \\ s(\neg A) &= \neg s(A) \\ s(A \vee B) &= s(A) \cup_L s(B) \\ s(A \wedge B) &= s(A) \cap_L s(B). \end{aligned}$$

The assertion or denial of the atomic formula A produces a new set-up. This is accomplished by the add function. The addition of atomic formula A with value v to set-up s is denoted by $\text{add}(A, v, s)$. This is defined as

$$\text{add}(A, v, s) = \{(p, v) \mid (p, v) \in s \wedge p \neq A\} \cup \{(A, s(A) \cup_L v)\}.$$

Since assertions of the form $A \vee B$, for example, cannot be accommodated within the

		$\neg A$	
		Both	Both
A	T	F	
	F	T	
	None	None	

		$A \vee B$			
		Both	T	F	None
A	Both	Both	T	Both	T
	T	T	T	T	T
	F	Both	T	F	None
	None	T	T	None	None

		$A \wedge B$			
		Both	T	F	None
A	Both	Both	Both	F	F
	T	Both	T	F	None
	F	F	F	F	F
	None	F	None	F	None

Figure 3

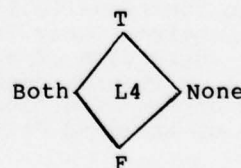


Figure 4

framework of a single set-up, it is necessary when allowing compound inputs to define a knowledge state to be a set of set-ups rather than a single one. A knowledge state E will thus be a non-empty set of set-ups. The add function is extended to sets of set-ups and compound inputs as follows

$$\begin{aligned} \text{add}(A, T, E) &= \{\text{add}(A, T, s) \mid s \in E\} \\ &\quad \text{when } A \text{ is atomic} \\ \text{add}(A, F, E) &= \{\text{add}(A, F, s) \mid s \in E\} \\ &\quad \text{when } A \text{ is atomic} \\ \text{add}(\neg A, T, E) &= \text{add}(A, F, E) \\ \text{add}(\neg A, F, E) &= \text{add}(A, T, E) \\ \text{add}(A \vee B, T, E) &= \text{add}(A, T, E) \cup \\ &\quad \text{add}(B, T, E) \\ \text{add}(A \vee B, F, E) &= \text{add}(B, F, \text{add}(A, F, E)) \\ \text{add}(A \wedge B, T, E) &= \text{add}(B, T, \text{add}(A, T, E)) \\ \text{add}(A \wedge B, F, E) &= \text{add}(A, F, E) \cup \\ &\quad \text{add}(B, F, E). \end{aligned}$$

Since each set-up element of a knowledge state may contain too much information to correctly answer a query, computation of the value of an expression from a knowledge state E uses the meet operation in the approximation lattice η_A :

$$E(A) = \eta_A(\{s(A) \mid s \in E\}).$$

A consequence of these definitions is that if A and $\neg A \vee B$ are both asserted with value T , the QAS will not report that B has value T as would be the case with conventional logic. (Instead, $(A \vee \neg A) \vee B$ has value T .) Thus, inferences by, for example, modus ponens cannot be accomplished by merely asserting the implication.

Belnap's four-valued logic has rule inputs of the form $A \rightarrow B$ to assert B whenever A is asserted. The knowledge state E is extended to an information state (R, E) with the addition of a set of rules R of the form $A \rightarrow B$ which can be invoked after each assertion.

The result of applying rule $A \rightarrow B$ to knowledge state E is

$$\cup\{\text{add}(B, s(A) \eta_A T, s) \mid s \in E\}.$$

In other words, B is asserted to be T in every set-up s of E for which $s(A)$ is at least T (is T or Both).

3. Fuzzy logic

Zadeh's developed fuzzy logic for dealing with uncertainty. Fuzzy logic uses real numbers in the range $[0,1]$ as truth values, the magnitude of the truth value representing the strength of or confidence in the assertion, 1 representing absolute truth and 0 representing absolute falsity. Dealing with inconsistency was not envisioned since the value of the negation of a proposition is mandated to be one minus the value of the proposition. Truth values for compound formulas are computed using the following rules.

$$s(\neg A) = 1 - s(A)$$

$$\begin{aligned} s(A \vee B) &= \max(s(A), s(B)) \\ s(A \wedge B) &= \min(s(A), s(B)) \end{aligned}$$

4. Fuzzy four-valued logic

Fuzzy four-valued logic generalizes each of Belnap's four-valued logic and Zadeh's fuzzy logic. Truth values are represented by points in the unit square, expressed as two element vectors $\langle x, y \rangle$. The first component, x , in the range $[0,1]$ expresses the strength of or confidence in the assertion of the proposition; the second component, y , in the range $[0,1]$ expresses the strength of or confidence in the denial of the proposition.

The two truth values of classical logic can be represented by $T = \langle 1, 0 \rangle$ and $F = \langle 0, 1 \rangle$. The added truth values of Belnap's four-valued logic can be represented by $\text{Both} = \langle 1, 1 \rangle$ and $\text{None} = \langle 0, 0 \rangle$. Truth values of fuzzy logic can be represented by values of the form $\langle x, 1-x \rangle$ where x is in the range $[0,1]$.

The operation of our logic is nearly identical to that of Belnap's four-valued logic. The truth values define an approximation lattice Aus with partial ordering \leq_A , join operation \cup_A , and meet operation \cap_A as follows

$$\begin{aligned} \langle x_1, y_1 \rangle &\leq_A \langle x_2, y_2 \rangle \\ &\quad \text{iff } x_1 \leq x_2 \text{ and } y_1 \leq y_2 \\ \langle x_1, y_1 \rangle \cup_A \langle x_2, y_2 \rangle &= \langle \max(x_1, x_2), \max(y_1, y_2) \rangle \\ \langle x_1, y_1 \rangle \cap_A \langle x_2, y_2 \rangle &= \langle \min(x_1, x_2), \min(y_1, y_2) \rangle. \end{aligned}$$

It is depicted in Figure 5 with only the extreme points corresponding to Belnap's four truth values shown.

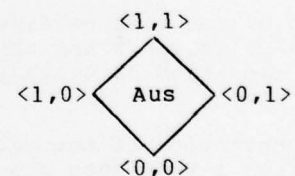


Figure 5

As in Belnap's four-valued logic, let a set-up s be represented as a set of ordered pairs with distinct first elements $(p, \langle x, y \rangle)$ of atomic formula p with value $\langle x, y \rangle$. Then the function s which computes truth values assigned by set-up s is defined recursively by

$$\begin{aligned} s(A) &= \text{the } \langle x, y \rangle \text{ such that } (A, \langle x, y \rangle) \in s \\ &\quad \text{when } A \text{ is atomic} \\ s(\neg A) &= \langle y, x \rangle \text{ where } s(A) \text{ is } \langle x, y \rangle \\ s(A \vee B) &= s(A) \cup_L s(B) \\ s(A \wedge B) &= s(A) \cap_L s(B). \end{aligned}$$

Here, \cup_L and \cap_L are computed in the logical lattice Lus (shown with extreme points only in Figure 6) as follows:

$$\begin{aligned}\langle x_1, y_1 \rangle \cup_L \langle x_2, y_2 \rangle &= \langle \max(x_1, x_2), \min(y_1, y_2) \rangle \\ \langle x_1, y_1 \rangle \cap_L \langle x_2, y_2 \rangle &= \langle \min(x_1, x_2), \max(y_1, y_2) \rangle.\end{aligned}$$

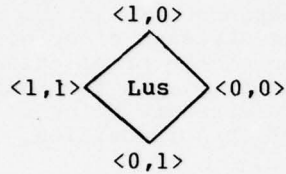


Figure 6

The assertion or denial of the atomic formula A produces a new set-up. The addition of atomic formula A with value $\langle x, y \rangle$ to set-up s is denoted by $\text{add}(A, \langle x, y \rangle, s)$. This is defined as

$$\begin{aligned}\text{add}(A, \langle x, y \rangle, s) &= \\ &= \{(p, v) \mid (p, v) \in s \wedge p \neq A\} \\ &\cup \{(A, s(A) \cup_A \langle x, y \rangle)\}.\end{aligned}$$

The add function is extended to sets of set-ups and compound inputs as follows

$$\begin{aligned}\text{add}(A, \langle x, y \rangle, E) &= \{\text{add}(A, \langle x, y \rangle, s) \mid s \in E\} \\ &\quad \text{when A is atomic} \\ \text{add}(\neg A, \langle x, y \rangle, E) &= \text{add}(A, \langle y, x \rangle, E) \\ \text{add}(A \vee B, \langle x, y \rangle, E) &= \\ &= \text{add}(A, \langle x, y \rangle, \text{add}(B, \langle 0, y \rangle, E)) \cup \\ &\quad \text{add}(A, \langle 0, y \rangle, \text{add}(B, \langle x, y \rangle, E)) \\ \text{add}(A \wedge B, \langle x, y \rangle, E) &= \\ &= \text{add}(A, \langle x, y \rangle, \text{add}(B, \langle x, 0 \rangle, E)) \cup \\ &\quad \text{add}(A, \langle x, 0 \rangle, \text{add}(B, \langle x, y \rangle, E)).\end{aligned}$$

Note that $\langle 0, y \rangle$ can also be expressed as $\langle x, y \rangle \cap_A \langle 0, 1 \rangle$ or $\langle x, y \rangle \cap_A F$ and that $\langle x, 0 \rangle$ can also be expressed as $\langle x, y \rangle \cap_A \langle 1, 0 \rangle$ or $\langle x, y \rangle \cap_A T$.

Again, computation of the value of an expression from a knowledge state E uses the meet operation in the approximation lattice \cap_A :

$$E(A) = \cap_A \{s(A) \mid s \in E\}.$$

Use of rules in fuzzy four-valued logic is also similar to that in Belnap's four-valued logic.

The result of applying rule A*B to knowledge state E is

$$\cup \{\text{add}(B, s(A) \cap_A T, s) \mid s \in E\}.$$

In other words, B is asserted to be $\langle x, 0 \rangle$ in every set-up s of E for which s(A) is $\langle x, y \rangle$ for some y.

Properties of fuzzy four-valued logic shared by Belnap's four-valued logic are:

- (1) minimality: $\text{add}(A, V, E)$ is the weakest possible operation which makes A have value at least V,
- (2) permanence: once $\text{add}(A, V, E)$ is performed A will forever have value at least V,
- (3) history independence: it doesn't matter in what order a set of assertions is performed.

Finally, we present a simple example of the reasoning capability of fuzzy four-valued logic. Suppose we assert $A \vee B$ with value $\langle x, y \rangle$ in the empty knowledge state:

$$\begin{aligned}\text{add}(A \vee B, \langle x, y \rangle, \{\}) &= \\ &= \{(A, \langle x, y \rangle), (B, \langle 0, y \rangle)\}, \\ &= \{(A, \langle 0, y \rangle), (B, \langle x, y \rangle)\}.\end{aligned}$$

Call the resulting knowledge state E. Then $E(A \vee B)$ is $\langle x, y \rangle$ and each of $E(A)$ and $E(B)$ is $\langle 0, y \rangle$ since neither has been asserted separately with any strength to be true and their disjunction has been denied with strength y.

5. References

- 1 Belnap, N. D. Jr. How a computer should think. Contemporary Aspects of Philosophy. Proceedings of the Oxford International Symposium, 1975, pp. 30-56.
- 2 Belnap, N. D. Jr. A useful four-valued logic. Proceedings of the 1975 International Symposium on Multiple-valued Logic, Bloomington, Indiana, p. 399 (summary).
- 3 Zadeh, L. A. Fuzzy sets. Information and Control 8 (1965), 338-353.

FOUR-VALUED THRESHOLD LOGIC FULL ADDER CIRCUIT IMPLEMENTATIONS

K. W. Current and D. A. Mow
Department of Electrical Engineering
University of California
Davis, CA 95616

Abstract

A four-valued logic full adder accepts two four-valued inputs and a binary carry input and produces a two-quaternary-digit, base-four, output word that is the sum of the values of the inputs. In this paper we present two new current-mode logic realizations of the four-valued logic full adder. Also reviewed is a recent I^2L realization of this function. In some applications, the use of these four-valued logic full adders requires fewer than 50% of the devices necessary to implement the same function with binary full adders.

Introduction

One advantage of multivalued logic over standard binary logic in the design of digital large-scale-integrated (LSI) circuits is its significantly greater functional density. With the trend today toward ever larger digital LSI circuit chips, fabrication limits are increasingly encountered. Production yield is strongly and inversely correlated with LSI circuit die area. Thus, for a given minimum acceptable processing yield in a particular semiconductor technology, there exists a maximum allowable die area. That factor which limits the amount of digital signal processing that can be accomplished on a chip of given maximum area is, in general, the coverage of the silicon surface with interconnecting metal signal lines, not the number of transistors and resistors. Since each four-valued variable may assume four logical states, twice the information carrying capacity as binary logic, a significant reduction in the total number of signal lines necessary to implement many logical functions is possible. The circuits presented in this paper require about 50% fewer transistors and resistors and signal interconnections than do their all-binary counterparts. Thus, multivalued logic's greater functional density could be used to take maximum advantage of limited chip area.

The use of multivalued logic in integrated injection logic (I^2L) circuits is now being explored commercially [1,2]. One new I^2L circuit developed for this application is the four-valued logic full adder. In the next section, we briefly summarize the operation of this circuit. In the third and fourth sections we present current-mode logic realizations of the four-valued logic full adder, and then compare binary and four-valued realizations of a 31-input counter.

Four-Valued I^2L Threshold Logic Full Adder

Some operations in digital signal processing are easily amenable to implementation with four-valued logic; for example, adding and counting. The four-valued I^2L threshold logic full adder [3,4] is under investigation because of its superior functional density and compatibility with conventional I^2L . A four-valued logic full adder adds two four-valued inputs A and B, and binary carry input C_i to yield the base-four sum as a two-digit word CS , where C is the most significant digit. The truth table for this operation is shown in Figure 1.

Since the circuits to be described implement threshold logic functions, let us assume the following definition for a general four-valued threshold function. A function $F(\underline{x})$ of n binary variables $\underline{x} = (x_1, x_2, \dots, x_n)$ is a threshold function if there exists a vector of integer weights $\underline{a} = (a_1, a_2, \dots, a_n)$ and thresholds $T_i \triangleq T_i - .5$, where T_i are integers, such that

$$F = \begin{cases} 3 & \text{if } \underline{a} \cdot \underline{x} = \sum_{j=1}^n a_j x_j > T_3 \\ 2 & \text{if } T_2 < \underline{a} \cdot \underline{x} < T_3 \\ 1 & \text{if } T_1 < \underline{a} \cdot \underline{x} < T_2 \\ 0 & \text{if } \underline{a} \cdot \underline{x} < T_1 \end{cases}$$

Four-valued logical inputs \underline{x} assume values ZERO, ONE, TWO, and THREE and are interpreted as weighted sums of binary inputs. The two basic operations of a threshold function are the formation of the weighted sum $\underline{a} \cdot \underline{x}$ and the comparison of that sum with the thresholds. The addition of two base-four numbers A and B and binary input carry C_i will produce a weighted sum within the range $0 \leq \underline{a} \cdot \underline{x} \leq 7$. Representing this weighted sum in base-four with the two-digit output CS requires the C output to assume only binary values (ZERO and ONE). This function is easily implemented in I^2L and current-mode threshold logic.

A four-valued I^2L threshold logic full adder [3] is shown in Figure 2. All logical variables are represented by easily generated and duplicated multiples of a unit value of current. The three inputs A, B, and C_i are duplicated by the three input current mirrors and summed in an analog fashion to yield two copies of the weighted sum

$a \cdot x$. One copy is converted from sunk current to sourced current via the pnp mirror at the top of the figure for use at the SUM output; the other is compared to the 3.5 threshold gate. If $a \cdot x$ is less than 3.5, the threshold gate remains on, the 4 units of constant current are diverted away from the SUM output device and the output is $SUM = a \cdot x$. At the same time the CARRY current output is ZERO. If $a \cdot x$ is greater than 3.5, the threshold detector will turn off and the 4 units of constant current are mirrored by the SUM output device. The SUM output current is now $a \cdot x - 4$, and the CARRY output becomes ONE. These devices have been fabricated and tested [3] and their results are apparently encouraging.

We estimate the propagation delay through a four-valued I^2L threshold logic full adder to be 50ns to 150ns based upon data in [3]. For faster operation, emitter-coupled-logic (ECL) compatible four-valued logic has been investigated. In the next section we describe a four-valued logic full adder that is easily developed from previously reported circuitry.

Basic Four-Valued CML Threshold Logic Full Adder

In the previous section, we saw how simply the I^2L threshold logic full adder operates. I^2L is basically a current-mode logic form. The generation of four-valued logic functions with I^2L amounts to duplicating, summing, differencing, and comparing amounts of current. ECL is also current-mode logic, so we may duplicate, sum, difference, and compare quantities of current using ECL-based circuitry and integrated circuit design techniques. ECL is the fastest standard logic family due to its nonsaturating operation. These speed advantages are sought in the four-valued current-mode threshold logic full adder circuit to be described next. Similar circuits were used in [5] to implement a parallel divider array.

Shown in Figure 3 is a block diagram of the four-valued threshold logic full adder. The idea is to decode the weighted sum $a \cdot x$ from decimal 0 through 7 to base-four 00 through 13. Thus, input $a \cdot x = 2$ would activate $SUM = "2"$ and $CARRY = 0$, and input $a \cdot x = 6$ would activate $SUM = "2"$ and $CARRY = 1$, for example. Detector "2" is a 2-out-of-7 detector in parallel with a 6-out-of-7 detector since the base-four output word's least significant digit (SUM) assumes a value of TWO when the input is either $a \cdot x = 2$ or $a \cdot x = 6$. Similarly, detectors "1" and "3" contain 1-out-of-7 and 5-out-of-7 and 3-out-of-7 and 7-out-of-7 detectors, respectively. The schematic for the "2" detector is shown in Figure 4.

Again, we are using quantities of current to represent the logic variables. The weighted sum $a \cdot x$ is formed as in the I^2L case; by the analog summing of logical currents. This sum is then converted into voltage $V = (a \cdot x) \cdot I \cdot R$ and dc level shifted to V_1 and V_2 as shown in the schematic. Let us assume that $I \cdot R = .4v$ and $V_D = .8v$ to simplify the explanation. When V_1 and V_2 are

are greater than V_{T_2} and V_{T_2}' (inputs $a \cdot x = 0,1$),

the $2I_0$ gate current is diverted to ground. When $a \cdot x = 2$, $V = -.8v$, $V_1 = -2.4v$, and $V_2 = -3.2v$. In this case, $V_2 > V_{T_2}'$ and $V_1 < V_{T_2}$, so $2I_0$ (a

logical TWO) appears on the SUM line. Likewise, a TWO SUM output appears when $V_2 > V_{T_2}'$ and $V_1 < V_{T_2}$.

The "1" and "3" detectors generate SUM outputs in the same way but with I_0 and $3I_0$ currents respectively. Since only one of the six threshold gates used to generate the SUM digit is "on" for any given input, they may be wired-OR. The carry circuit is merely a differential pair with a threshold voltage set to deliver I_0 of current to the output line for an $a \cdot x = 4$ input. Notice that the SUM and CARRY output currents may be direct coupled to other full adders of this type.

Simulations on SPICE2 [6] using data on high speed ECL transistors [7] indicate propagation delays of about 5ns for this implementation of a four-valued logic full adder. One drawback to this circuit technique is the need for a power supply of about 6.6 volts or more. ECL compatible, binary outputs may be easily generated from the four-valued output variables with additional decoding m-out-of-7 detectors. This conversion and another current-mode logic realization of the four-valued logic full adder are discussed in the next section.

Switching Threshold Four-Valued CML Full Adder

Another circuit implementation of the four-valued threshold logic full adder is now presented. This approach again uses current-mode logic but develops the SUM and CARRY outputs in an entirely different manner. Its power supply requirement has been improved to 6 volts. The unique feature of this circuit is the way in which the threshold reference voltages are developed. Transfer characteristics are shown in Figure 5. Two sets of threshold reference voltages are possible, one for the input range $0 \leq a \cdot x \leq 3$ and another for $4 \leq a \cdot x \leq 7$. As $a \cdot x$ increases from ZERO to THREE, a unit of current is directed to a summing node developing outputs 0, 1, 2I, and 3I. When $a \cdot x$ reaches 4, the base-four number 10 must be displayed. A secondary reference circuit is switched with $a \cdot x = 4$ to turn back off all three threshold circuits previously turned on for $a \cdot x$ inputs 1, 2, and 3. At these new threshold reference voltages, inputs $a \cdot x = 5, 6, 7$ will turn on those same three gates and the SUM output will again be 1, 2I and 3I. Thus, the same devices are used to perform two completely separate thresholding tasks.

A circuit schematic of the four-valued threshold logic full adder is shown in Figure 6A, and threshold reference voltage circuit is in Figure 6B. The weighted sum input $(a \cdot x) \cdot I$ is passed through resistor R to develop a proportional voltage, $V = (a \cdot x)IR$. Transistors T_1 , D_1 and D_2 act as an emitter follower and level shifter whose output, V_1 , is the input to four

differential pairs that generate the SUM and CARRY outputs, and to the threshold reference voltage circuit. Voltages V_{RS1} , V_{RS2} , V_{RS3} , and V_{RC} are

threshold reference voltages for the generation of logical ONE, TWO, and THREE in the SUM circuit, and a logical ONE in the CARRY circuit, respectively. With $a \cdot x$ less than 3.5, V_1 is greater than V_{RC} and current I_A in Figure 6B will be directed to V_C ; when greater than 3.5, V_1 is less than V_{RC} and I_A is directed through resistor R_A in the reference generation leg of the circuit. Each reference voltage is then shifted down $I_A R_A$ volts, enough to reset SUM to ZERO, as this threshold is exceeded.

As V_1 moves through its range of values corresponding to inputs ZERO through SEVEN, transistors T_4 , T_6 , and T_8 in Figure 6A direct unit values of current into the summing node of the SUM bit as the ONE, TWO, and THREE thresholds are exceeded. As the FOUR threshold is exceeded, transistor T_{10} diverts a unit of current into the CARRY output and simultaneously the thresholds fall to their new values turning off transistors T_4 , T_6 , and T_8 , forcing the SUM bit to ZERO. As V_1 continues to decrease through values for FIVE through SEVEN, the SUM output assumes values ONE, TWO and THREE. Thus, with a minimum of components we can perform the four-valued logic full add operation. Current sources may be implemented with any simple standard technique such as that shown in Figure 6C.

This circuit was also simulated and its worst case propagation delay is shown in Figure 6D to be about 10ns; a significant improvement over I^2L .

To be compatible with ECL, each four-valued current variable must be converted to its two-bit binary equivalent voltage. A four-valued current to binary voltage converter is shown in Figure 7A. It converts a four-valued SUM current to voltage and then that voltage is input to 1-out-of-3 and 3-out-of-3 threshold detectors which generate binary ONE'S when the four-valued variable is ONE or THREE. The voltage outputs of these detectors are OR'ed with emitter followers. This is the binary 2^0 bit. A current switch is thresholded such that it provides a binary ZERO output for the 2^1 bit given a four-valued ZERO or ONE, and a 2^1 bit ONE output for four-valued inputs TWO and THREE. Since the CARRY output is already binary 2^2 , it need only be converted from current to ECL compatible voltage swing and level shifted as shown in Figure 7B.

Four-valued Full Adder Application

For the purpose of comparison, let us describe an application of four-valued logic full adders to the design of a parallel counter. A parallel counter is a multiple input circuit that counts the number of ONE'S on its inputs. A 31-input parallel counter using binary ECL full adders is shown in Figure 8. This same function implemented with four-valued logic full adders is shown in Figure 9. The binary version requires 936 transistors and resistors while the four-valued logic version requires 480 transistors and resistors; a

significant reduction. The binary version could easily run at 70 MHz while the four-valued one would operate at about 23 MHz. Power requirements are about the same.

Conclusion

Four-valued threshold logic full adder circuit implementations have been presented. Two are designed to be ECL input and output level and swing compatible. They require NPN transistors and resistors only and are thus compatible with all standard bipolar technologies. Significant savings in devices are possible with four-valued logic in some applications as has been shown. It is hoped that this greater functional density of four-valued logic will motivate its further study and gradual introduction into use.

Acknowledgment

Work supported in part by intercampus grant 3-5225-61900-5.

References

1. T. T. Dao, "Electronics Newsletter," *Electronics*, p. 26, Oct. 14, 1976.
2. "Four-level logic," *Electronics*, pp. 31-32, Oct. 28, 1976.
3. T. T. Dao, "Threshold I^2L and its applications to binary symmetric functions and multivalued logic," *IEEE Journal on Solid-State Circuits*, vol. SC-12, pp. 463-472, Oct. 1977.
4. N. Freidman, et al., "Realization of multivalued integrated injection logic full adder," *IEEE Journal on Solid-State Circuits*, vol. SC-12, pp. 532-534, Oct. 1977.
5. J. K. Newton, "An implementation of the Stephanelli multivalued parallel divider array," 6th Internat. Symp. on Multiple Valued Logic Proc. pp. 61-67, May 1976.
6. L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Ph.D. dissertation, University of California, Berkeley, May 1975.
7. A. E. Cosand, "A very high-speed, low-power bipolar integrated circuit process," 1973 *IEDM Digest*, pp. 35-37, 1973.

		SUM				CARRY			
B	A	0	1	2	3	0	1	2	3
	0	0	1	2	3	0	0	0	0
$C_i = 0$	1	1	2	3	0	0	0	0	1
	2	2	3	0	1	0	0	1	1
	3	3	0	1	2	0	1	1	1
	0	1	2	3	0	0	0	0	1
$C_i = 1$	1	2	3	0	1	0	0	1	1
	2	3	0	1	2	0	1	1	1
	3	0	1	2	3	1	1	1	1
	0	1	2	3	0	0	0	0	1

Fig. 1. Four-valued Logic Full Adder Truth Table

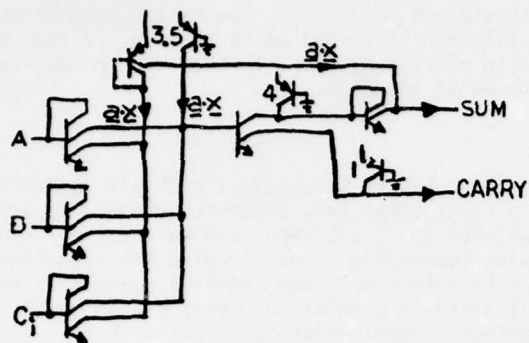


Fig. 2. Four-valued I^2L Threshold Logic Full Adder

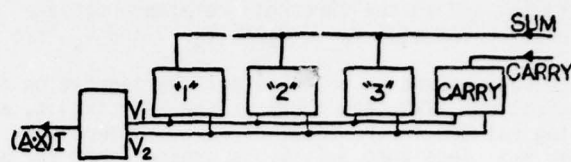


Fig. 3. Four-valued CML Threshold Logic Full Adder Block Diagram

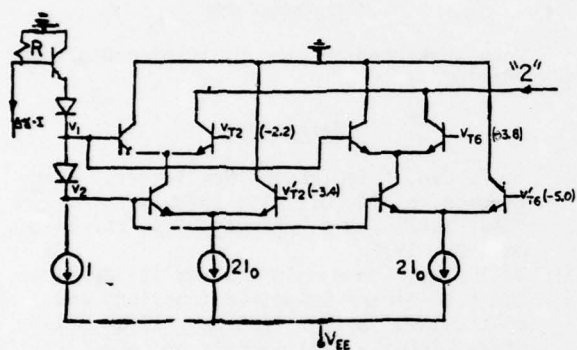


Fig. 4. CML Threshold Logic "2" Detector

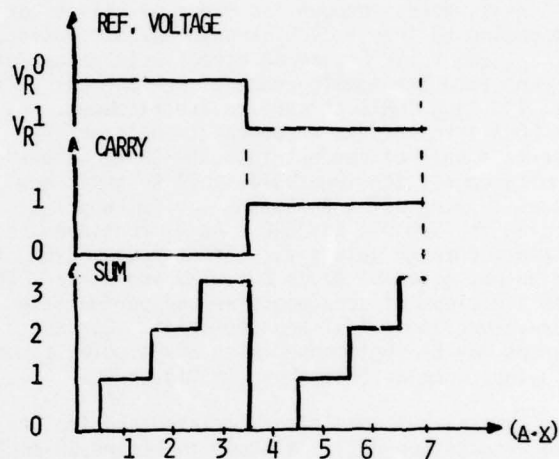


Fig. 5. Switched Threshold Full Adder Transfer Characteristic

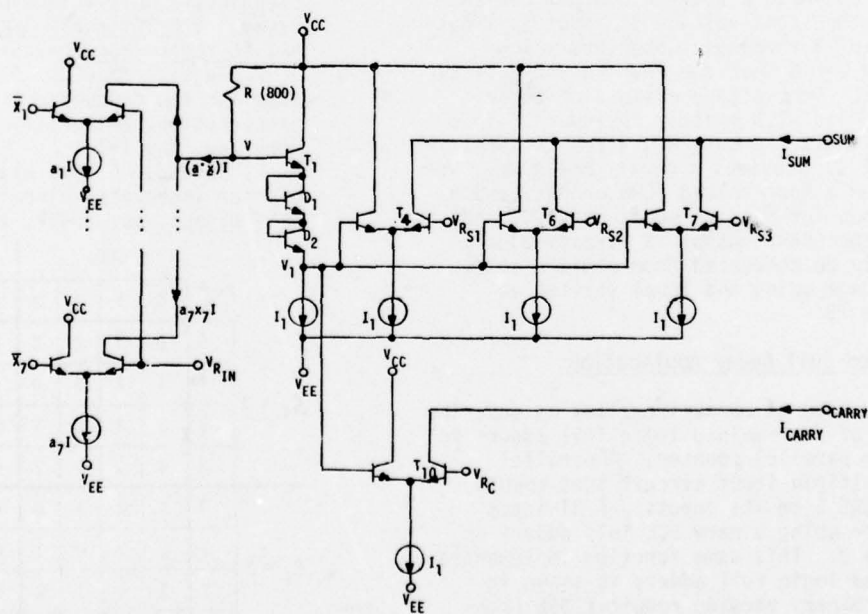


Fig. 6A. Four-valued Switched Threshold Logic Full Adder

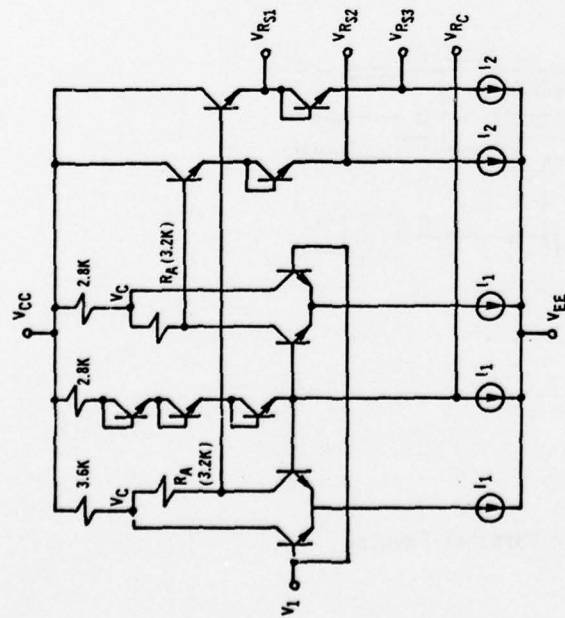


Fig. 6B. Threshold Reference Voltage Circuit

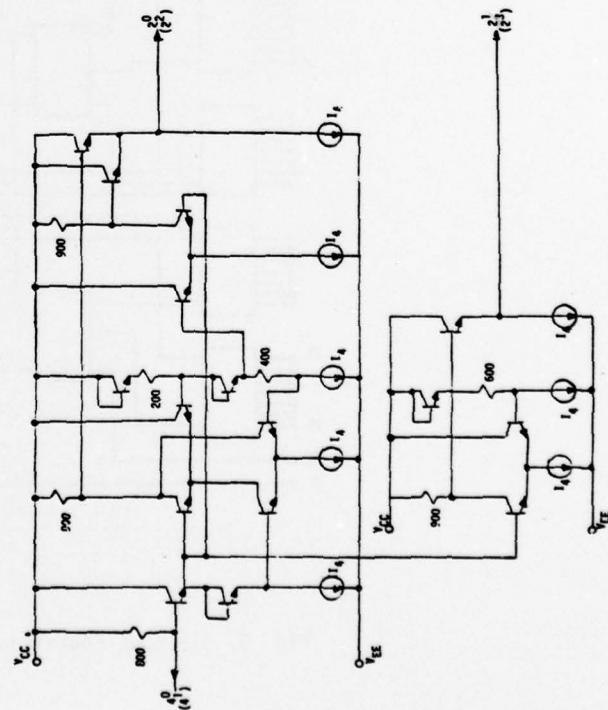


Fig. 7A. Four-valued SUM Current to Binary Voltage Converter

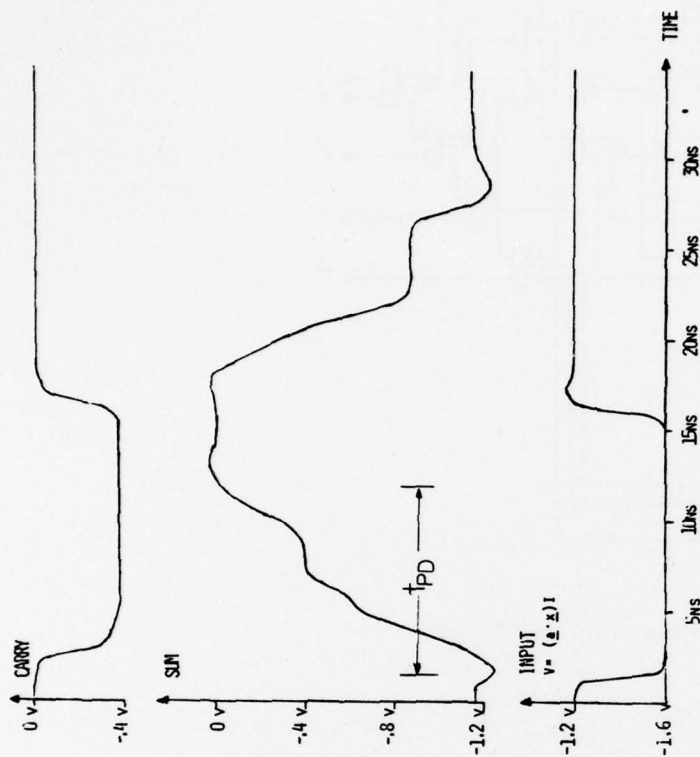


Fig. 6D. Worst Case Propagation Delay

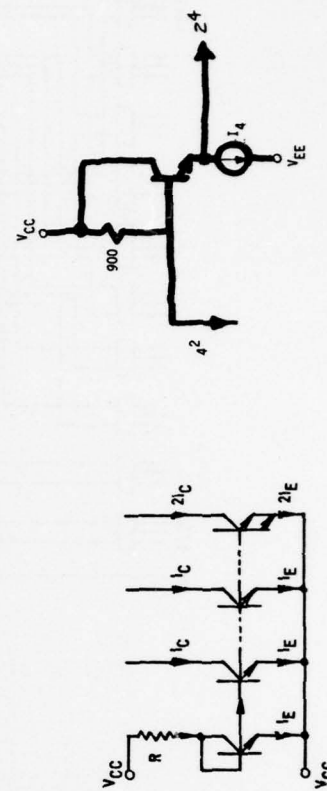


Fig. 6C. Simple Current Source

Fig. 7B. CARRY CONVERTER

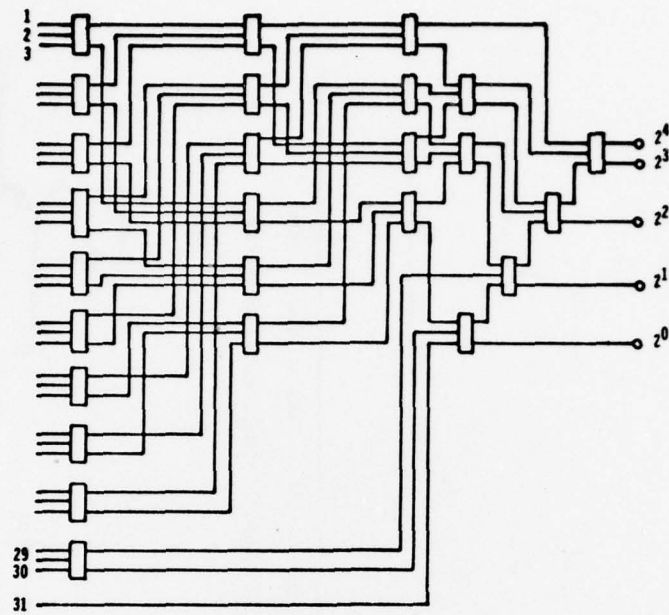


Fig. 8. 31-Input Binary Parallel Counter

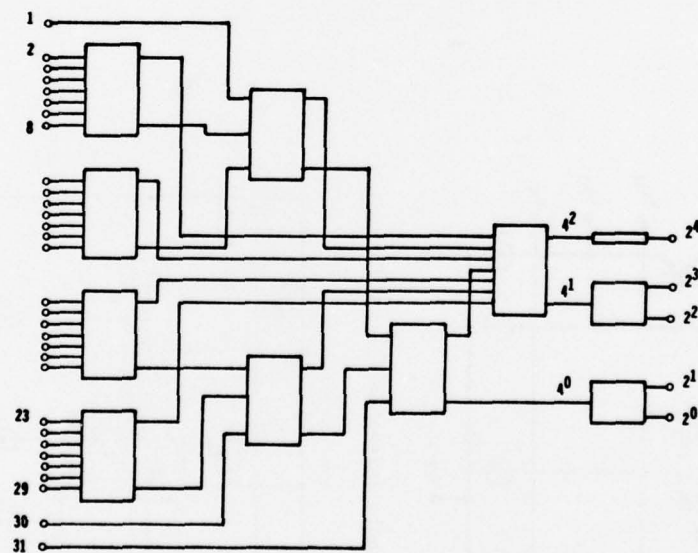


Fig. 9. 31-Input Four-Valued Logic Parallel Counter

THE MODULAR COMPLEXITY OF A TREE STRUCTURED HIGHER RADIX MULTIPLIER*

James R. Armstrong
Assistant Professor
Department of Electrical Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

Abstract

The modular complexity of a tree structured higher radix multiplier is analyzed. The number of required modules is shown to be closely approximated

by $2n^2(1 + \frac{1}{R-1}) + (\frac{R-3}{R-1})n$ where R is the radix n is the number of positions in the numbers to be multiplied.

I. Introduction

In this paper the modular complexity of a higher radix multiplier is analyzed. The multiplier is a simultaneous multiplier, i.e., a multiplier implementation employing shifting and adding. Simultaneous multiplication is the fundamental method of multiplication since at some point in the shifting and adding process, a simultaneous multiplier circuit is employed to multiply a certain number of multiplier positions by the multiplicand.

Simultaneous multiplication has previously been shown to be of $O(n^2)$, [1] i.e. if n is the number positions in the multiplier and multiplicand, and N is the number of modules required to implement the circuit then:

(1) $N \approx Cn^2$ as n becomes large. Then since $n = \lceil \log_R K \rceil$, (where K is the largest integer to be represented by n positions) and $\log_R K = \ln K / \ln R$, we have:

$$(2) N \approx C'(\ln K)^2 / (\ln R)^2$$

and one is able to determine the effect of radix on modular complexity. In this paper we analyze a tree structure higher radix simultaneous multiplier and show that for this circuit, C' is $2 + \frac{2}{R-1}$, i.e. a function of radix. Thus the radix effects not only input size n , but also the modular complexity of the circuit itself.

We interpret the term modular complexity as being the number of modules required to mechanize a given function. From a graph theoretic point of view we are determining the number of "interior nodes" in a network necessary to connect "input nodes" to "output nodes." We do not consider the complexity of the modules themselves (although this is a primary concern of research in the field of multi-

valued logic) but only their number. The reader should also note that we use the term circuit to designate a network of modules.

II. Multiplier Structure

One can form the simultaneous product of two, n position, "R-ary" numbers, by performing the following operations:

- 1) Generation of position products and product carries.²
- 2) Formation of the sum of the position by position products and any carries from the previous stage.
- 3) Generation of "sum carries" from the position by position products and any carries from the previous stage.

As a simple example, let $X = A_1 + A_2R$ and $Y = B_1 + B_2R$, then the product of X and Y is a 4 position product, i.e.

$$P_1 = A_1B_1$$

$$P_2 = A_2B_1 + A_1B_2 + C_{p11}$$

$$P_3 = A_2B_2 + C_{p21} + C_{p12} + C_{s2}$$

$$P_4 = C_{p22} + C_{s3}$$

where C_{sx} and C_{pxx} are sum and product carries respectively. Note that any product or sum carries are collected at the next stage for summation with the position by position products. Figure 1 below shows a mechanization in terms of the modules required. There are four basic types of modules involved. Product (P) modules produce the low order digit of a position by position product, while the product carry (C_p) modules produce the high order digit of the product. For example, if $A_1 = 9$ and $B_1 = 6$, and the arithmetic is in base 10, then $P_1 = 4$ and $C_{p11} = 5$. At each position, a modulo R adder (\oplus) sums the products and carries from the previous position. A sum carry (C_s) circuit receives the same inputs as the modulo R adder and generates carries for the next position.

Since the carries (including the sum carries) are collected at the next stage, there is a necessary restriction on the number of inputs a sum carry circuit can have. In order to generate a carry which does not go beyond the next stage we must have no more than $R + 1$ inputs into the circuit. This is true since each input to the carry circuit has a maximum value of $R - 1$, and $(R + 1) \cdot (R - 1) = R^2 - 1$, which is the maximum value that one can have without generating a carry beyond the next

²For radices higher than 2 only.

*This research was supported by the National Science Foundation under Grant No. ENG 76-09925.

$\lceil X \rceil$ denotes the smallest integer greater than or equal to X .

position. In the case where the number of inputs exceeds $R + 1$, more than one carry circuit is required. (In Figure 1, this situation does not arise, since no C_s circuit has more than four inputs).

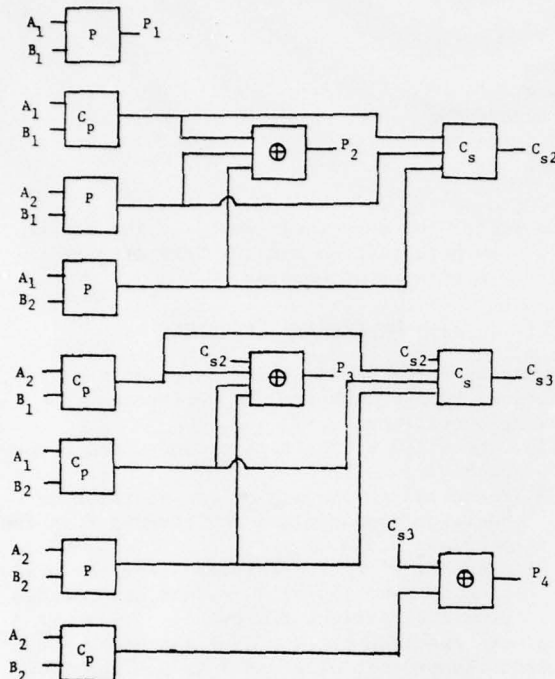


Figure 1

Figure 2 is a generalized diagram of the i th multiplier stage of an R -ary multiplier. As in the first example, product (P) and product carry modules (C_p) generate the product and product carry respectively. The input carries from the previous stage and the product module outputs for this stage are inputs to a tree of R -ary full adders. Each full adder has $R + 1$ inputs and produces the sum modulo R and a possible carry to the next stage. (The R -ary full adder thus combines the function of the sum carry module (C_s) and the modulo R adder shown in the first example). Some of the $R + 1$ adders may have unused inputs but this inefficiency is more than compensated for by the fact that all modules are identical.

If the number of inputs into the tree is n_i , then the number of full adder modules in the tree is given by [2]:

$$(3) \quad N_i = \left\lceil \frac{n_i - 1}{R + 1 - 1} \right\rceil = \left\lceil \frac{n_i - 1}{R} \right\rceil$$

and since

$$(4) \quad \begin{aligned} n_1 &= 0 \\ n_i &= 2i - 1 + N_{i-1} \\ 2 &\leq i \leq n \end{aligned}$$

$$\text{and} \quad n_i = 4n - 2i + 1 + N_{i-1}$$

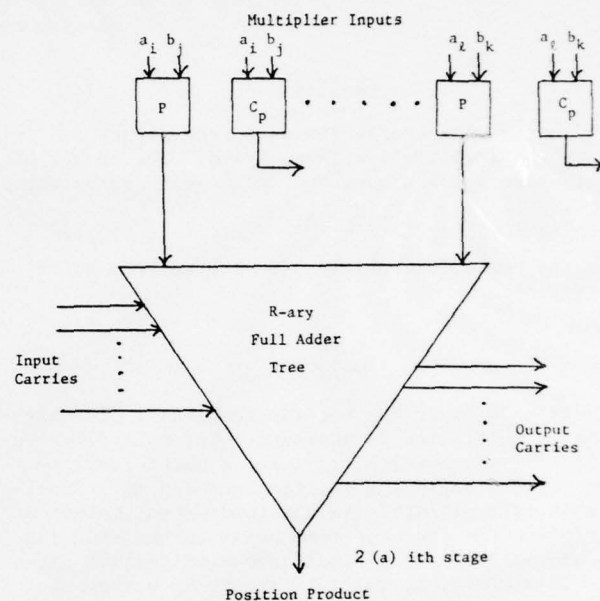
$$(5) \quad n+1 \leq i \leq 2n$$

The total number of modules in all required trees can be evaluated recursively and summed over $2n$.

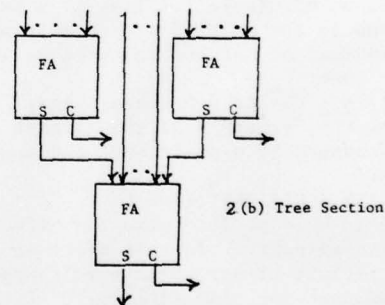
The number of product and product carry modules is $2n^2$ (independent of radix) and therefore, the total number of modules for the n position d -ary multiplier is given by:

$$(6) \quad N = 2n^2 + \sum_{i=1}^{2n} N_i$$

Thus, the total number of modules in the circuit can be calculated. What is needed, however, is a closed form expression for N , which exhibits the relationship between N , n , and R .



2 (a) i th stage



2 (b) Tree Section

Figure 2. i th Multiplier Stage

III. Closed Form Approximation for N

A first step in obtaining a closed form approximation to N is to solve the recurrence relations:

$$(7) \quad n_i = 2i - 1 + \left\lceil \frac{n_{i-1} - 1}{R} \right\rceil \text{ for } 2 \leq i \leq n$$

$$(8) \quad n_i = 4n - 2i + 1 + \left\lceil \frac{n_{i-1} - 1}{R} \right\rceil \text{ for } n+1 \leq i \leq 2n$$

A direct solution of these equations would appear to be inhibited by the presence of the term $\left\lceil \frac{n_{i-1} - 1}{R} \right\rceil$.

However, consider the recurrence relation obtained from (7) and (8) by dropping the least integer greater than brackets, i.e.,

$$(9) x_i = 2i - 1 + \frac{x_{i-1} - 1}{R} \text{ for } 2 \leq i \leq n$$

$$(10) x_i = 4n - 2i + 1 + \frac{x_{i-1} - 1}{R} \text{ for } n+1 \leq i \leq 2n$$

The solution to these equations will yield more than an approximation, a fact which is demonstrated by:

Lemma 1

$$n_i = \lceil x_i \rceil$$

Proof:

$$x_i = 2i - 1 + (x_{i-1} - 1)/R$$

$$\lceil x_i \rceil = \lceil 2i - 1 + (x_{i-1} - 1)/R \rceil$$

$$= 2i - 1 + \left\lceil \frac{x_{i-1} - 1}{R} \right\rceil$$

and since

$x_1 = n_1 = 1$, the lemma is true. a similar argument can be made for the other recurrence relation. The solutions to (9) and (10) are:

(for $R \geq 3$)

$$(11) x_i = \frac{2R^2}{(R-1)^2 R^i} + \frac{2Ri}{(R-1)} + \frac{(-R^2 - 2R + 1)}{(R-1)^2} \text{ for } 2 \leq i \leq n$$

$$(12) x_i = \frac{-2(R+1)R^{n+1}}{(R-1)^2 R^i} - \frac{2Ri}{R-1} + \frac{(4n+1)R^2 - 4nR + 1}{(R-1)^2}$$

$$\text{for } n+1 \leq i \leq 2n$$

The exact expression for N is given by:

$$(13) N = \sum_{i=1}^{2n} \left\lceil \frac{x_i - 1}{R} \right\rceil + 2n^2$$

A first approximation to N is given by:

$$(14) N^* = \sum_{i=1}^{2n} \frac{x_i - 1}{R} + 2n^2$$

Using the two recurrence equation solutions given by (11) and (12) one gets:

$$(15) N^* = \frac{2R}{(R-1)^3} + (2n^2 - 2n)/(R-1) + 2n^2$$

If one assumed that the error incurred in dropping the $\lceil \cdot \rceil$ from (13) results in an average error of .5 for each term in the summation, then the total error for 2n terms is given by n. If we add this term to N^* we get

$$(16) \tilde{N} = \frac{2R}{(R-1)^3} + \frac{2n^2}{R-1} + \frac{n(R-3)}{R-1} + 2n^2$$

Computer evaluation of \tilde{N} and N for $3 \leq R \leq 10$ and $2 \leq n \leq 10$ shows the average error between \tilde{N} and N to be .4, while the worst case error is 1.75.

A further simplification can be made if it is noted that for $3 \leq R$, the term $2R/(R-1)^3$ is less than one and decreases rapidly toward zero as R increases:

$\therefore 2n^2/R-1 + \frac{n(R-3)}{R-1} + 2n^2$ is a good approximation to N.

Finally for large n, the n^2 term predominates and one gets the form stated in the introduction, i.e.

$$N \approx \frac{(\ln K)^2}{(\ln R)^2} \left(2 + \frac{2}{R-1} \right)$$

Summary

An expression for the modular complexity of a tree structured higher radix multiplier has been presented. This expression demonstrated that the radix effects the complexity of the circuit not only through the input size (i.e. $n = \lceil \ln K / \ln R \rceil$), but also through its effect on the modular structure of the circuit itself. This second effect is demonstrated by the $2n^2/R-1 + n(R-3)/R-1$ term in (16). This effect is caused by the limitation on the number of inputs that an R-ary full adder can have (R+1) without generating a carry beyond the next position.

The above analysis gives the complexity of the modular net without regard to the complexity of the module themselves. The circuit is made up of three module types, i.e.:

n^2 product modules

n^2 product carry modules

$\frac{2n^2}{R-1} + \frac{n(R-3)}{R-1}$ R-ary adders

The next step in the complexity analysis is to assume complexity functions for the modules (vs. radix) and thus determine the overall complexity of the circuit as a function of R. If N is the modular net complexity, and C_m the complexity of the modules, then the overall circuit complexity (C_N) is given by:

$$C_N = NC_m$$

An interesting question to consider then is: for a given function, what types of C_m will allow C_N to be a monotonically decreasing function of radix? This question is the concern of the writer's current research effort and will be discussed in a future paper.

References

- 1) Ofman, Yu, "On the Algorithmic Complexity of Discrete Functions", Dokl, Akad, Nauk SSR, Vol. 145, No. 1, 1962, pp. 48-51.
- 2) Armstrong, J. R. "The Efficient Single Output (d,r) Circuit", Eighth Annual Southeastern Symposium on System Theory, April 1976 pp. 63-66.
- 3) Armstrong, J. R. "The Relationship Between Computational Circuit Complexity and Radix", Proceedings of the Sixth International Symposium on Multivalued Logic, May 1976, pp. 175-178.

ADDITION IN SIGNED DIGIT NUMBER SYSTEMS

M. DAVIO and J.-P. DESCHAMPS

MBLE RESEARCH LAB. BRUSSELS.

ABSTRACT.

The paper studies a general class of signed number representation systems. Among the members of that class, it selects the number representation systems that allow one to build multi-summand ripple carry-like adders enjoying simultaneously the properties of saturation, expandability and symmetry.

1. INTRODUCTION.

The study presented in this paper has a two-fold motivation : it first starts from the increased need of custom design large scale integrated circuits such as multipliers, product accumulators, dividers,.... which find interesting applications wherever the universal or general purpose architectures fail to meet some system requirement, most often the speed.

Next, it tries to use the characteristics of multivalued logics with all its possible consequences on circuit design such as :

- (i) increase of information density over the connections which allows one to improve the "efficiency" of these connections and, as the case may be, to overcome pin limitations.
- (ii) description of algorithms endowed with an increased intrinsic parallelism. Such algorithms are obviously candidate to provide in any logics (including the binary logics) interesting alternative designs.

It is however difficult to achieve these two goals simultaneously. An example will help to clarify this point. Think of a ripple carry adder for two summands. If we may expect from technological advances that two full-adders operating

e.g. in the bases 2 and 4 have similar delay times, the time performance of the base 4 adder will be about twice better than that of the binary adder so that, if speed is the design goal, one would incline to the quaternary adder. However, in this case, the carry remains binary and in a quaternary implementation no increase of the carry connection efficiency would result. And, as we shall see, this type of problem becomes even more critical if one turns to the problem adding two signed numbers in the usual numeration systems (b's complement notation, negative bases).

The present paper tries to approach that type of problem in the case of multi-summand ripple carry adders. The emphasis is placed on three properties that we may intuitively describe here as follows :

- (i) *saturation* or maximal exploitation of the freedom degrees allowed to the circuit connections
- (ii) *expandability* of the basic cell to adders of arbitrary length.
- (iii) *symmetry* of the input carry with the input digits which implies simpler designs.

It has been found that the classical methods used to represent signed numbers fail to meet (at least simultaneously) the above three criteria. This is why a larger class of number representations defined in section 2 as signed digit representations has been investigated. The essential result, described in section 4 is the description of number representation systems in which the above three properties exist simultaneously.

2. SIGNED DIGIT NUMBER REPRESENTATION SYSTEMS.

2.1. Preliminary remarks.

Classical arithmetic makes a frequent use of *positional* or *weighted number representation systems*. An integer A is represented by a vector \underline{a} with integer components :

$$A \sim [a_{n-1}, \dots, a_1, a_0] \quad (1)$$

A weight w_i , generally an integer, is attached to each component of the representation. The number A and its representation are related by :

$$A = \sum_{i=0}^{n-1} a_i w_i \quad (2)$$

If the components a_i and the weights w_i all are nonnegative integers, the interval R_n only contains nonnegative integers. In such a situation, one uses a sign to distinguish positive and negative integers and one speaks of *sign and magnitude representation*. The heterogeneous nature of such a representation occasionally causes trouble in arithmetic operations, as it is the case when adding two oppositely signed numbers.

This is the reason why other types of representations have been investigated. In particular, one observes that the set R_n may contain negative members if some digits a_i and/or some weights w_i have negative values. In what follows, we shall always assume that the weights w_i are positive integers, while the digits a_i may assume positive and negative values. We shall show in section 2.3 that this is the most general situation and that it in particular covers most of the irredundant number systems in practical use. We have thus reached in this way the concept of signed digit number systems. A class of signed digit number systems is introduced in section 2.2. It should be mentioned here, to prevent any confusion, that other signed digit number systems have been investigated in the past!^{2 3 4}

A mixed-radix signed-digit number representation system is defined by two vectors \underline{b} and \underline{k} with integer components :

$$\underline{k} = [k_{n-1}, \dots, k_1, k_0]; \quad 0 \leq k_i < b_i, \quad i. \quad (4)$$

We shall furthermore use the vector

$$\underline{r} = [r_{n-1}, \dots, r_1, r_0] \quad (5)$$

defined by :

$$r_i = -(b_i - k_i - 1) ; \forall i \quad (6)$$

From the \underline{b} -vector, we now define the weight vector :

$$\underline{w} = [w_n, w_{n-1}, \dots, w_1, w_0] \quad (7)$$

by :

$$w_0 = 1 \quad (8)$$

$$w_i = w_{i-1} b_{i-1} = \prod_{j=0}^{i-1} b_j; \quad i \in \{1, 2, \dots, n\} \quad (9)$$

From the latter definition of the w_i 's, one immediately infers that the weights satisfy the useful identity

$$\sum_{i=0}^{k-1} (b_i - 1)w_i = w_k - 1. \quad (10)$$

The main property is described by theorem 1.

Theorem 1. Any integer A belonging to the interval :

$$R_n = \left[\sum_{i=0}^{n-1} r_i w_i, \sum_{i=0}^{n-1} k_i w_i \right] \quad (11)$$

has a unique representation :

$$\underline{a} = [a_{n-1}, \dots, a_1, a_0] \quad (12)$$

such that :

$$(i) \quad r_i \leq a_i \leq k_i; \quad i \in \{0, 1, \dots, n-1\} \quad (13)$$

$$(ii) \quad A = \sum_{i=0}^{n-1} a_i w_i \quad (14)$$

Proof. The proof essentially rests upon the following set of equations :

$$A = q_1 b_0 + a_0 \quad ; \quad r_0 \leq a_0 \leq k_0 \quad (15_0)$$

$$q_1 = q_2 b_1 + a_1 \quad ; \quad r_1 \leq a_1 \leq k_1 \quad (15_1)$$

$$q_i = q_{i+1} b_i + a_i; \quad r_i \leq a_i \leq k_i \quad (15_i)$$

$$q_{n-1} = q_{n-1}^b + a_{n-1}; \quad r_{n-1} \leq a_{n-1} \leq k_{n-1}. \quad (15_{n-1})$$

These equations uniquely define the consecutive q_i 's and the a_i 's satisfying (13), since the interval $[r_i, k_i]$ is a complete residue system modulo b_i . By successive substitutions, one obtains from (15) :

$$A = q_n w_n + \sum_{i=0}^{n-1} a_i w_i. \quad (16)$$

The proof is thus completed if we show that $q_n = 0$. Assume e.g. that $q_n \geq 1$. Then, by (16) and (13); we obtain :

$$A \geq w_n + \sum_{i=0}^{n-1} r_i w_i = w_n - \sum_{i=0}^{n-1} (b_i - k_i - 1) w_i ,$$

i.e., if we use (10)

$$A \geq 1 + \sum_{i=0}^{n-1} k_i w_i ,$$

and thus contradicts the hypothesis $A \in R_n$, with R_n given by (11). Similarly, one deduces that, if $q_n \leq -1$, the integer A is smaller than the lower bound of R_n . \square

Example 1. Let $\underline{b} = [5, 4, 3, 2]$ and $\underline{k} = [2, 2, 1, 1]$. Using upper bars to denote negative digits, we have: $\underline{r} = [\bar{2}, 1, \bar{1}, 0]$ and $\underline{w} = [120, 24, 6, 2, 1]$. Furthermore, $R_4 = [-56, 63]$. Let us e.g. compute the representation of -43 (decimal) :

$$\begin{aligned} -43 &= (-22) \cdot 2 + (1) ; & 0 \leq a_0 \leq 1 \\ -22 &= (-7) \cdot 3 + (-1) ; & -1 \leq a_1 \leq 1 \\ -7 &= (-2) \cdot 4 + (1) ; & -1 \leq a_2 \leq 2 \\ -2 &= (0) \cdot 5 + (-2) ; & -2 \leq a_3 \leq 2 . \end{aligned}$$

Hence $-43 = (-2) \cdot (24) + (1) \cdot (6) + (-1) \cdot (2) + (1) \cdot (1)$. The decimal -43 is represented as $[2, 1, \bar{1}, 1]$ in our number system. \square

2.3. Particular cases.

2.3.1. Signed weight systems.

A signed digit system becomes a *signed weight system* iff :

$$\forall i, \text{ either } k_i = b_i - 1 \text{ or } k_i = 0 . \quad (17)$$

Thus, in a signed weight system :

$$\forall i, \text{ either } 0 \leq a_i \leq b_i - 1 \text{ or } -(b_i - 1) \leq a_i \leq 0 . \quad (18)$$

In this case, we may observe that :

(i) the vector \underline{k} may be replaced by a binary vector \underline{s} . We adopt here for \underline{s} the following definition:

$$s_i = 0 \text{ iff } k_i = b_i - 1 ; s_i = 1 \text{ iff } k_i = 0 . \quad (19)$$

(ii) the positions in which negative digits appear are fixed, once for ever throughout the system. It is thus clear from (14) that, in such a case, the negative signs may be transferred from the a_i 's to the corresponding w_i 's according to :

$$a_i' = (-1)^{s_i} a_i ; w_i' = (-1)^{s_i} w_i . \quad (20)$$

In this case, all the a_i 's are positive while (14) is replaced by :

$$A = \sum_{i=0}^{n-1} a_i' w_i' . \quad (21)$$

This is the reason why we call these systems signed weight systems.

Remarks

(i) If $\underline{s} = \underline{0}$ (the all-zero vector), the signed weight systems reduce to the classical *mixed radix systems*. An everyday life example of such a system is given by $\underline{b} = [24, 60, 60]$ used for time measurements in days, hours, minutes, seconds. The mixed radix systems are also currently used in residue arithmetic^{5,6,7,8}.

(ii) An alternative definition of a signed weight number system may be formulated in terms of a single vector

$$\tilde{\underline{b}} = [\tilde{b}_{n-1}, \dots, \tilde{b}_1, \tilde{b}_0] ; |\tilde{b}_i| > 1 \quad (22)$$

of signed integers. Such a definition is built from the vectors \underline{b} and \underline{s} by :

$$t_0 = s_0 \quad (23)$$

$$t_i = s_{i-1} \oplus s_i ; i \in \{1, 2, \dots, n-1\} \quad (24)$$

$$\tilde{b}_i = (-1)^{t_i} b_i . \quad (25)$$

Conversely, given a vector \underline{b} such that

$$\tilde{b}_i = (-1)^{t_i} |\tilde{b}_i| , \quad (26)$$

one defines \underline{b} and \underline{s} vectors by :

$$b_i = |\tilde{b}_i| \quad (27)$$

$$s_i = \bigoplus_{j=0}^i t_j \quad (28)$$

2.3.2. Homogeneous number systems.

A signed digit system is an *homogeneous number system* iff $b_i = b, \forall i$. Number representation systems which have received practical application are most often both signed weight and homogeneous; we shall briefly review some of the most striking examples.

(i) positive integer representation. Whenever $\underline{s} = \underline{0}$, one obtains, for $b = 2, 8, 10, 16, \dots$ the classical binary, octal, decimal, hexadecimal, ... representations of positive integers.

(ii) b's complement representation. Here

$$\underline{s} = [1, 0, 0, \dots, 0] , \quad (29)$$

i.e. the most significant digit is interpreted with a negative weight $-b^{n-1}$. The representation interval is given by :

$$R_n = [-(b-1)b^{n-1}, b^{n-1}-1] \quad (30)$$

and the integer A is related to its representation \underline{a} by :

$$A = -a'_{n-1} b^{n-1} + \sum_{i=0}^{n-2} a'_i b^i. \quad (31)$$

When $b=2$, one obtains the celebrated two's complement representation. In the general case $b > 2$, the domain of a'_{n-1} is sometimes restricted to $\{0,1\}$, so as to replace R_n by another (smaller) interval symmetric about 0. See⁹.

(iii) negative base representation. One uses here

$$\underline{a} = [(n-1)_{\text{mod } 2}, \dots, 1, 0, 1, 0]. \quad (32)$$

In this case, an integer A is related to its representation by

$$A = \sum_{i=0}^{n-1} a'_i (-b)^i. \quad (33)$$

The latter equation explains the term "negative base" used for these systems. Note that A is positive if its representation has an odd number of significant digits and negative in the opposite case. An easy summation shows furthermore that the representation interval is given by :

$$R_n = \left[-\frac{b^{n+1}-b}{b+1}, \frac{b^n-1}{b+1} \right]; (n \text{ even}) \quad (34_0)$$

$$R_n = \left[-\frac{b^n-b}{b+1}, \frac{b^{n+1}-1}{b+1} \right]; (n \text{ odd}) \quad (34_1)$$

Additional literature on these systems (mostly in the case $b=2$) may be found in ^{10,11,12}. A relationship between negative bases and complex number representations is pointed out in ^{13,8}.

2.3.3. Fully homogeneous number systems.

A signed digit number representation is *fully homogeneous* iff $b_i=b$ and $k_i=k$ for all i .

In particular, a fully homogeneous system is *centered* iff $k = \lfloor (b-1)/2 \rfloor$ ($\lfloor x \rfloor$ represents the integer part of x). Here

$$R_n = \left[-\frac{b}{2} \cdot \frac{b^n-1}{b-1}, \left(\frac{b}{2}-1\right) \frac{b^n-1}{b-1} \right]; (b \text{ even}) \quad (37_0)$$

$$R_n = \left[-\frac{b^n-1}{2}, \frac{b^n-1}{2} \right]; (b \text{ odd}). \quad (37_1)$$

These systems enjoy interesting properties : one notes here the symmetry of the representation interval. Other interesting properties of these systems will appear later on. The "balanced ternary

system" ($b=3, k=1$) is described in ⁸.

3. BASE CONVERSION

3.1. Problem statement; general solutions.

The base conversion problem may be stated as follows : given the representation $\underline{a}^{(1)}$ of A in a first number representation system called *source system*, obtain the representation $\underline{a}^{(2)}$ of A in a second number representation system called *target system*. In what follows, we assume that the source and target systems are defined by the vector pairs $(\underline{b}^{(1)}, \underline{k}^{(1)})$ and $(\underline{b}^{(2)}, \underline{k}^{(2)})$ respectively. We furthermore assume that the number A has a representation in both systems.

Two basic methods are available for performing a base conversion. They differ by the number system in which the calculations take place. If the calculations are to be performed in the source system, the digits of the target representation will be obtained according to (15) as the remainders of a series of integer divisions. This has already been illustrated by example 1 and we shall not comment that method further on.

If the calculations are to be performed in the target system, a simple method may be based on the familiar formula :

$$A = \sum_{i=0}^{n-1} a'_i {}^{(1)} w_i {}^{(1)} \quad (38)$$

A simple way of putting formula (38) at work consists in using its Hörner scheme form :

$$A = (((\dots (a'_{n-1} {}^{(1)} b_{n-2} {}^{(1)} + a'_{n-2} {}^{(1)} b_{n-3} {}^{(1)} + \dots + a'_2 {}^{(1)}) b_1 {}^{(1)} + a'_1 {}^{(1)}) b_0 {}^{(1)} + a'_0 {}^{(1)}). \quad (39)$$

The computation is thus performed in $(n-1)$ steps each of which involves the multiplication by $b_i {}^{(1)}$ and the addition of $a'_i {}^{(1)}$. That computation is possible as soon as :

- (i) the representations of the $a'_i {}^{(1)}$ in the target system are known.
- (ii) it is possible to multiply by $b_i {}^{(1)}$ and to add in the target system.

The first of these two requirements explains why that method is most often suggested in homogeneous systems when $b^{(1)} < b^{(2)}$. In this case, the digits $a'_i {}^{(1)}$ are their own representation in the target system. The argument is however by no means compulsory.

Example 2. Convert [1,12,3,57] seconds expressed in the day, hours, minutes, seconds system into its decimal equivalent :

$$((1.24 + 12)60 + 3) 60 + 57 = 129837 \text{ seconds. } \square$$

Remark. It is obviously possible to speed up the computation involved by (39) by using an evaluation scheme of a more parallel type. This is illustrated by figure 1 for $n=8$. It is clear that the underlying construction may be extended to all n and that the conversion time is then proportional to $\log n$. The new algorithm however uses more complex operations than the above mentioned Hörner scheme.

3.2. Particular cases.

3.2.1. Homogeneous systems.

In the case of homogeneous systems, the Hörner scheme (39) becomes an iteration. If we denote by $b^{(1)}$ and $b^{(2)}$ the source and target bases, and by R a partial result, we may describe the typical iteration step by :

$$R := R b^{(1)} + s. \quad (40)$$

Our purpose is to show that, for homogeneous systems, the iteration step (40) may itself be described as a more refined iteration. Let $[r_{n-1}, \dots, r_1, r_0]$ denote the representation of the partial result in the target basis. There exists a set of q_i 's such that :

$$\begin{aligned} R &= q_1 b^{(2)} + r_0 \\ q_1 &= q_2 b^{(2)} + r_1 \\ &\vdots \\ q_{n-1} &= 0 \cdot b^{(2)} + r_{n-1}. \end{aligned} \quad (41)$$

Consider now the following equations (43) that uniquely define the integers q_i^* and r_i^* under the condition

$$r_i^{(2)} \leq r_i^* \leq k_i^{(2)}. \quad (42)$$

$$\begin{aligned} r_0 b^{(1)} + s &= q_1^* b^{(2)} + r_0^* \\ r_1 b^{(1)} + q_1^* &= q_2^* b^{(2)} + r_1^* \\ &\vdots \\ r_{n-1} b^{(1)} + q_{n-1}^* &= q_n^* b^{(2)} + r_{n-1}^* \\ q_n^* &= q_{n+1}^* b^{(2)} + r_n^* \\ &\vdots \\ q_{m-1}^* &= 0 \cdot b^{(2)} + r_{m-1}^* \end{aligned} \quad (43)$$

If one next substitutes the values (43) into the system (41) first multiplied by $b^{(1)}$, one obtains the new system :

$$\begin{aligned} b^{(1)} R + s &= (q_1 b^{(1)} + q_1^*) b^{(2)} + r_0^* \\ q_1 b^{(1)} + q_1^* &= (q_2 b^{(1)} + q_2^*) b^{(2)} + r_1^* \\ &\vdots \\ q_{n-1} b^{(1)} + q_{n-1}^* &= q_n^* b^{(2)} + r_{n-1}^* \\ &\vdots \\ q_{m-1}^* &= 0 \cdot b^{(2)} + r_{m-1}^*, \end{aligned} \quad (44)$$

which shows that the vector $[r_{m-1}^*, \dots, r_1^*, r_0^*]$ actually is the expected representation of $b^{(1)} R + s$ in the target system. The importance of equations (43) becomes clear in the light of the above discussion. They indeed resolve the computation (40) in an iteration, whose typical step is described by :

$$r_i b^{(1)} + q_i^* = q_{i+1}^* b^{(2)} + r_i^* \quad (45)$$

A particular case of the above transformation is described in ¹⁴.

Example 3. Figure 2b illustrates the conversion of $A \sim [\bar{1} \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1]$ (2's complement) to $A \sim [\bar{1} \ 3 \ \bar{2} \ \bar{2} \ \bar{3} \ 1]$ (radix -4). Each cell performs (figure 2a) :

$$2r_i + q_i^* = 4q_{i+1}^* + r_i^*$$

by selecting $r_i \in \{0, 1, 2, 3\}$ for i even and $r_i \in \{0, -1, -2, -3\}$ for i odd. \square

3.2.2. Translations.

A base conversion is a *translation* if the source and target basis vectors are identical. Thus $\underline{b}^{(1)} = \underline{b}^{(2)} = \underline{b}$. In this case, the representation domains $R_n^{(1)}$ and $R_n^{(2)}$ have the same cardinalities and $R_n^{(2)}$ thus results from $R_n^{(1)}$ by a translation along the axis. That type of base conversion is particularly simple and deserves a specific study.

Let indeed

$$A = \sum_{i=0}^{n-1} a_i^{(1)} w_i; \quad r_i^{(1)} \leq a_i^{(1)} \leq k_i^{(1)}. \quad (46)$$

We see that A may be written under the form :

$$A = \sum_{i=0}^{n-1} (a_i^{(1)} + k_i^{(2)} - k_i^{(1)}) w_i - \sum_{i=0}^{n-1} (k_i^{(2)} - k_i^{(1)}) w_i. \quad (47)$$

The latter expression allows us to interpret A as the difference

$$A = X - Y \quad (48)$$

of two integers X and Y written in the target system. Indeed, one easily deduces from (46)

$$r_i^{(2)} \leq a_i^{(1)} + k_i^{(2)} - k_i^{(1)} \leq k_i^{(2)}, \quad (50)$$

while the defining inequalities

$$0 \leq k_i^{(1)} \leq b_i - 1 \text{ and } 0 \leq k_i^{(2)} \leq b_i - 1 \text{ easily yield}$$

$$r_i^{(2)} \leq k_i^{(2)} - k_i^{(1)} \leq k_i^{(2)}.$$

The process may be summarized as follows :

- (i) Given $\underline{a}^{(1)}$ form $\underline{x} : x_i = a_i^{(1)} + k_i^{(2)} - k_i^{(1)}$
- (ii) Form $\underline{y} : y_i = k_i^{(2)} - k_i^{(1)}$
- (iii) Subtract \underline{y} from \underline{x} in the target system.

Example 4 Convert

$$\underline{a}^{(1)} = [\bar{1} \ 0 \ \bar{2} \ 3 \ \bar{2} \ 0 \ \bar{2} \ 2] \text{ (radix } -4)$$

into 4's complement notation. Here :

$$\begin{aligned} \underline{k}^{(1)} &= [3 \ 0 \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \ 3] \\ \underline{k}^{(2)} &= [0 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3] \\ \underline{y} &= [\bar{3} \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \ 3 \ 0] \\ \underline{x} &= [\bar{3} \ 3 \ 0 \ 3 \ 0 \ 2 \ 0 \ 1 \ 3 \ 1 \ 0 \ 1 \ 2]. \end{aligned}$$

The final subtraction is :

$$\begin{array}{r} \text{carries} \quad \bar{1} \ \bar{1} \ \bar{1} \ \bar{1} \ \bar{1} \ \bar{1} \ \bar{1} \ \bar{1} \ \bar{1} \\ \underline{x} \quad \quad \bar{3} \ 3 \ 0 \ 3 \ 0 \ 2 \ 0 \ 1 \ 3 \ 1 \ 0 \ 1 \ 2 \\ \underline{y} \quad \quad \bar{3} \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \\ \hline \underline{a}^{(2)} \quad \bar{1} \ 3 \ 3 \ 3 \ 3 \ 2 \ 3 \ 2 \ 2 \ 1 \ 3 \ 2 \ 2 \end{array}$$

□

Remarks.

- (i) The above process may clearly be reverted to allow the computations to be performed in the source system.
- (ii) It is possible to present an equivalent algorithm as a flowchart, i.e. as a purely sequential process at each step of which a new digit of the target representation is obtained : the quantity \underline{y} and the transformation $\underline{a} \rightarrow \underline{x}$ do not appear explicitly in such an algorithm but are imbedded in the successive steps of the computation. Examples of that type of algorithm may be found in^{15,16,17,18}.

(iii) We also note that the operation of sign change immediately reduces to a translation. Indeed, if $[a_{n-1}, \dots, a_1, a_0]$ is the representation of some integer A in a $(\underline{b}, \underline{k})$ -number system, then, $[-a_{n-1}, \dots, -a_1, -a_0]$ is the representation of -A the number system $(\underline{b}, \underline{k}')$ where $k_i' = b_i - k_i - 1$. Let us e.g. apply the method to the sign change in b'_s complement notation. Here :

$$\underline{k} = \underline{k}^{(2)} = [0, b-1, \dots, b-1]$$

$$\underline{k}' = \underline{k}^{(1)} = [b-1, 0, \dots, 0].$$

Hence

$$\underline{y} = \underline{k}^{(2)} - \underline{k}^{(1)} = [-(b-1), (b-1), \dots, (b-1)],$$

i.e.

$$-Y = b^n - 2b^{n-1} + 1; \quad -Y \equiv -2b^{n-1} + 1 \pmod{b^n}$$

Similarly, it is easily observed that the number \underline{x} is obtained by replacing each digit of A by the complement to (b-1) of its absolute value while keeping signs unchanged. It may be checked that of -A has a representation, then $a_{n-1} \in \{0, -1\}$ and that the summation X-Y may be carried out modulo b^n to yield the correct result.

(iv) Let us finally call *slicing* the operation that consists in grouping the digits of a representation in blocks of equal length. That process is familiar when passing from binary to octal or to hexadecimal. It is noteworthy that the class of signed digit number systems is closed under the operation of slicing, i.e. that from any signed digit system, one obtains by any slicing a new signed digit system. The reader will easily convince himself that the restricted class of signed weight number systems does not enjoy that stability.

4. ADDITION.

4.1. Simple adders. Description and discussion.

We consider the problem of adding μ integers $X^{(j)}$, whose $(\underline{b}, \underline{k})$ -representations are given by :

$$\underline{x}^{(j)} \sim [x_{n-1}^{(j)}, \dots, x_1^{(j)}, x_0^{(j)}].$$

More precisely, if q_0 denotes an input carry, we wish to compute

$$A = q_0 + \sum_{j=1}^{\mu} x^{(j)}. \quad (49)$$

In this introductory subsection, we discuss the possibility of computing the result A by an algorithm similar to the one encountered in the ripple carry adder. In that algorithm, one

would thus compute, for each digit position the sum of digits in that position

$$S_i = \sum_{j=1}^{\mu} x_i^{(j)}, \quad (50)$$

add that sum to an appropriate input carry q_i and generate from $S_i + q_i$ a new carry q_{i+1} and a result digit a_i .

In what follows, we show that, if q_{i+1} and a_i are defined by :

$$\begin{aligned} S_i + q_i &= q_{i+1} b_i + a_i; \quad r_i \leq a_i \leq k_i \\ i &\in \{0, 1, \dots, n-1\} \end{aligned} \quad (51_i)$$

the result A is represented by $[q_n, a_{n-1}, \dots, a_1, a_0]$ in the (b, k) -number system.

Indeed, on the one hand, if we multiply each of the equations (51_i) by the corresponding weight w_i and if we sum these equations, we obtain:

$$q_0 + \sum_{i=0}^{n-1} w_i S_i = q_n w_n + \sum_{i=0}^{n-1} a_i w_i. \quad (52)$$

On the other hand, we obtain, from (49) and (50) :

$$A = q_0 + \sum_{j=1}^{\mu} \sum_{i=0}^{n-1} x_i^{(j)} w_i = q_0 + \sum_{i=0}^{n-1} w_i S_i, \quad (53)$$

and the comparison of (52) and (53) completes the proof of the assertion.

Equation (51_i) , which describes the typical step of the addition algorithm, becomes the central object of our discussion. A first remark is that equation (51_i) has no interest for design purposes, unless the ranges of the successive q_i 's are known and coherently defined. We shall assume :

$$r_i^* \leq q_i \leq k_i^* \quad (54)$$

and define, by symmetry

$$b_i^* = k_i^* - r_i^* + 1. \quad (55)$$

Now, the set of values I_i assumed by the left hand member of (51_i) will be called *input set* ; it is the interval :

$$I_i = [r_i^* + \mu r_i, k_i^* + \mu k_i]. \quad (56)$$

Similarly, the set of values O_i assumed by the right hand member of (51_i) will be called *output set* ; it is the interval

$$O_i = [r_{i+1}^* b_i + r_i, k_{i+1}^* b_i + k_i]. \quad (57)$$

With these notations, we may express as

$$I_i \subseteq O_i \quad (58)$$

the condition under which equation (51_i) may be considered as the behavioral description of a hardware adder cell. When an adder is built along these conditions, we call it a *simple adder*.

It is clear that, in any practical design situation, we would try to keep the successive carry ranges b_i^* as small as possible. In such a situation, the explicit version of (58), expressed as :

$$r_{i+1}^* b_i + r_i \leq r_i^* + \mu r_i \quad (59)$$

$$k_{i+1}^* b_i + k_i \geq k_i^* + \mu k_i \quad (60)$$

would allow one to choose recursively the r_i^* 's and k_i^* 's by :

$$r_{i+1}^* = \left\lceil \frac{r_i^* + (\mu-1)r_i}{b_i} \right\rceil \quad (61)$$

$$k_{i+1}^* = \left\lceil \frac{r_i^* + (\mu-1)k_i}{b_i} \right\rceil \quad (62)$$

where $\lceil x \rceil$ is the smallest integer not smaller than x .

4.2. Saturation.

4.2.1. General discussion.

We say that a simple adder is *saturated* iff condition (58) is replaced by

$$I_i = O_i.$$

Intuitively, this means that the cell connections are used with the maximum efficiency : stated otherwise, this would correspond to the best possible use of the multivalued character of the logic system.

In a saturated adder, the conditions (59) and (60) are replaced by any two of the following three equalities :

$$r_{i+1}^* b_i + r_i = r_i^* + \mu r_i \quad (64)$$

$$k_{i+1}^* b_i + k_i = k_i^* + \mu k_i \quad (65)$$

$$(b_{i+1}^* - \mu) b_i = b_i^* - \mu. \quad (66)$$

and the consecutive r_i^* 's and k_i^* 's should be obtained from (61) and (62) with no round off. Intuitively, we understand that the initial values

r_0^* and k_0^* must be chosen so as to satisfy that integrity condition. As a matter of fact, a recurrent use of (64) (65) and (66) immediately yields:

Lemma 1. *A simple adder enjoys the saturation property iff two of the following three conditions are satisfied for every $j=1,2,\dots,n$:*

$$r_0^* + (\mu-1) \sum_{s=0}^{j-1} r_s w_s = r_j^* w_j \quad (67)$$

$$k_0^* + (\mu-1) \sum_{s=0}^{j-1} k_s w_s = k_j^* w_j \quad (68)$$

$$b_0^* - \mu = (b_j^* - \mu) w_j \quad (69)$$

We now reach the main theorem of this section.

Theorem 2. *There exists a simple adder with the saturation property iff two of the following congruences are satisfied:*

$$r_0^* \equiv -(\mu-1) \sum_{s=0}^{n-1} r_s w_s \pmod{w_n} \quad (70)$$

$$k_0^* \equiv -(\mu-1) \sum_{s=0}^{n-1} k_s w_s \pmod{w_n} \quad (71)$$

$$b_0^* \equiv \mu \pmod{w_n} \quad (72)$$

Proof. The necessity is clear: it suffices to consider the equations (67)(68) and (69) with $j=n$ and to impose the integer character of the involved parameters. To prove the sufficiency, we observe that:

$$\alpha \equiv \beta \pmod{w_n} \Rightarrow \alpha \equiv \beta \pmod{w_j}.$$

Hence, if e.g. (72) holds true, (69) defines integer b_j^* 's. Similarly, if (70) and (71) hold true, (67) and (68) will define integer r_j^* 's and k_j^* 's. \square

Remark. When applying theorem 2, it is important to identify the quantities:

$$\sum_{s=0}^{n-1} r_s w_s \quad \text{and} \quad \sum_{s=0}^{n-1} k_s w_s$$

as the bounds of R_n as computed in section 2.

4.2.2. Examples.

(i) Fully homogeneous number systems.

In a fully homogeneous number system, conditions (70) and (71) become

$$r_0^* \equiv -\frac{(\mu-1)r}{(b-1)} \pmod{b^n} \quad (73)$$

$$k_0^* \equiv -\frac{(\mu-1)k}{(b-1)} \pmod{b^n} \quad (74)$$

In particular, if $(b-1)$ divides $(\mu-1)r$, the above conditions reduce to:

$$r_0^* \equiv \frac{(\mu-1)r}{(b-1)} \pmod{b^n} \quad (75)$$

$$k_0^* \equiv \frac{(\mu-1)k}{(b-1)} \pmod{b^n} \quad (76)$$

(ii) Negative bases

The discussion of the congruences (70) and (71) is slightly longer in the case of negative bases. That discussion will not be reproduced here. The reader will however check that the values

$$r_0^* = \sigma b \quad (77)$$

$$k_0^* = -\sigma \quad (78)$$

$$\sigma < 0 \quad (79)$$

$$\mu = -\sigma(b+1)+1 \quad (80)$$

define saturated simple adders for negative bases. As a matter of fact, it may be shown¹⁸ that conditions (77)...(80) are the most general conditions of saturation that are independent of n .

4.3. Expandability.

The concept of expandability is intuitively familiar: in the usual binary representation of integers, the full-adder is considered as expandable since it allows one to build adders for any size of the summands. In the case of simple adders, under study in section 4, expandability first requires the underlying algorithm to be an iteration; it next imposes the possibility of choosing

$$r_{i+1}^* = r_i^* \quad \text{and} \quad k_{i+1}^* = k_i^*. \quad (81)$$

Let us for example discuss expandability in terms of fully homogeneous systems. In this case, the coherence conditions (50) and (60) become:

$$r_{i+1}^* b + r \leq r_i^* + \mu r \quad (82)$$

$$k_{i+1}^* b + k \geq k_i^* + \mu k \quad (83)$$

Combining (81), (82) and (83), one obtains:

Theorem 3. *In a fully homogeneous number system, there exists an expandable simple adder for each couple of integers (r^*, k^*) satisfying*

$$r^* < \frac{(\mu-1)r}{(b-1)} \quad (84)$$

$$k^* > \frac{(\mu-1)k}{(b-1)} \quad (85)$$

It is interesting to discuss the existence of adders that are both saturated and expandable. The essential result is the following one :

Theorem 4. *In a fully homogeneous number system, there exists a simple adder which is both saturated and expandable iff*

$$(\mu-1)r/(b-1) \text{ and } (\mu-1)k/(b-1)$$

are integers.

Proof. We only prove the sufficiency. Indeed, if the conditions of the theorem are satisfied, the choice

$$r^* = \frac{(\mu-1)r}{b-1} ; k^* = \frac{(\mu-1)k}{b-1} \quad (86)$$

ensures saturation, by (75) and (76) and expandability, by (84) and (85). For a proof of the necessity see¹⁸ . □

5. CONCLUSION

In the course of our discussion, we progressively passed from the general class of signed digit number systems to the particular class of fully homogeneous systems. The latter are indeed the only ones with the help of which it is possible to build adders enjoying simultaneously the following three properties :

- (i) saturation (see (75) and (76))
- (ii) expandability (see (84) and (85))
- (iii) symmetry : we mean here that the input carry has exactly the same role to play as any of the other digits. This is clearly seen by setting $\mu=b$ in (86). The fact that the output functions are totally symmetric functions of their input variables allows one to apply to their synthesis appropriate simplifying methods¹⁹ .

No other signed number representation system simultaneously enjoys these three properties. For example, in negative bases, the systems described by (77)...(80) are never symmetric, while they may be shown to be expandable.

Fully homogeneous simple adders constructed with $b=\mu$ have an additional property, important from the hardware point of view : their design is independent of k . Indeed, the adder cell equation

$$q_i + x_i^{(1)} + \dots + x_i^{(\mu)} = q_{i+1}\mu + a_i$$

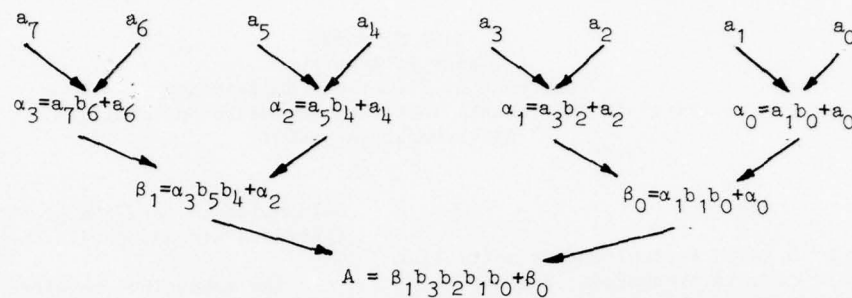
remains invariant if all the summands in both members are reduced by r .

Acknowledgement. The authors are indebted to Miss G. Gossart (formerly student, UCL) for pointing out the possibility of an independent handling of the expandability.

REFERENCES.

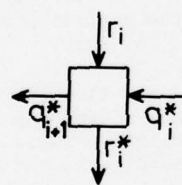
- ¹ Lehman M., Highspeed multiplication. IRE.TEC. EC-6, 3, pp. 204-205, 1957
- ² Booth A. D., A signed binary multiplication technique. Quart. J. Mech. and appl. math., 4, part 2, p.236, 1951.
- ³ Davio M. and Bioul G., Fast parallel multiplication. Philips Res. Repts, 32, pp. 43-70, 1977.
- ⁴ Avizienis A., Signed digit number representation for fast parallel arithmetic. IRE TEC, EC-10, 3, pp. 389-399, 1951.
- ⁵ Flores I., The Logic of Computer Arithmetic. Prentice Hall. Englewood Cliffs, N.J., 1963.
- ⁶ Szabo N. S. & Tanaka R. I., Residue arithmetic and its application to computer technology, Mc Graw Hill, N.Y., 1967.
- ⁷ Banerji D. K. , Residue arithmetic in computer design. Ph. D. Thesis. Univ. of Waterloo, Ontario, 1971.
- ⁸ Knuth D. E., The art of computer programming. vol.2, Seminumerical Algorithms. Addison Wesley. Reading Mass. 1969.
- ⁹ Loomis H. H., Data representation . In "Introduction to computer architecture (H. Stone Ed.) S. R.A. Chicago 1975.
- ¹⁰ Shannon C. E., A symmetrical notation of number. Amer. Math. Month., 57, p.90, 1950.
- ¹¹ Wadel L. B., Negative base number systems. IEEE TEC, EC-6, p.123, June 1957.
- ¹² Agrawal D. P., Some aspects of fast arithmetic computing circuits. Ph. D. Thesis, nr. 206, Dept. of Electrical Engineering, E.P.F. Lausanne, 1975.
- ¹³ Knuth D. E., An imaginary number system. Comm. ACM, 3, pp. 245-247, 1960.
- ¹⁴ Lanning W. C., General algorithms for direct radix conversion. Computer Design, pp. 61-73, June 1975.
- ¹⁵ De Regt M. P., Negative radix arithmetic Computer Design, May-Dec. 1967, Jan. 1968.
- ¹⁶ Zohar S., Negative radix conversion IEEE TC, C-19, March 1970.
- ¹⁷ Lanning W. C., Negative radix transformation algorithms, Information and Control, 34, pp. 271-285, 1977.
- ¹⁸ Davio M., Deschamps J.P., Addition in signed digit number systems, MBLE Research Lab., Rept. R361, Nov. 1977.

- 19 Davio M., Deschamps J.P., Thayse A., Discrete and Switching functions. Mc Graw Hill, To appear, spring 1978.

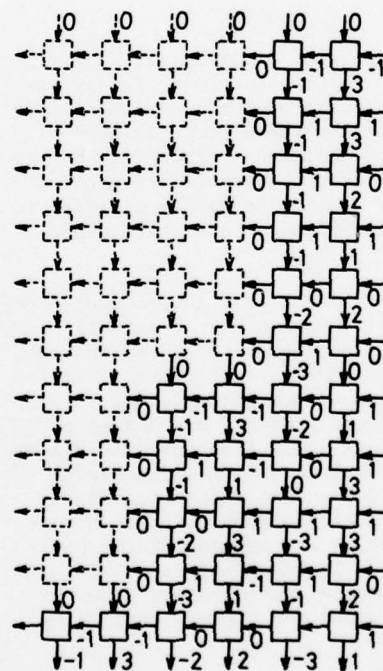


Logarithmic time base conversion

Figure 1.



a)



b)

Base conversion in homogeneous systems

FIGURE 2.

A SIMULTANEOUS, RADIX FOUR, I^2L MULTIPLIER MECHANIZED VIA REPEATED ADDITION*

Adit D. Singh
James R. Armstrong
Department of Electrical Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

Abstract

The design of a radix 4 simultaneous multiplier, utilizing I^2L circuits, is presented. Because of the relative efficiency of threshold logic implementations in I^2L to those using logic primitives, the multiplication is implemented by adding together copies of the multiplicand. This is shown to be more efficient than generating and summing single position partial products. The radix 4 design is also compared to an equivalent radix 2 multiplier (also mechanized with I^2L circuits), and is shown to be faster and less costly in terms of required chip area.

1. Introduction

The recent announcement by Signetics [1] of a functionally complete multivalued logic family has made higher radix design more of a practical possibility. These current mode circuits employ I^2L threshold logic to implement a four valued logic system. The circuit techniques are general with respect to radix and can be easily modified to realize either a three-valued or a higher-valued system [2], [3].

The advantages of the reduced interconnect complexity in higher radix systems have been well established. Circuit speed and complexity are the two other important factors to be studied in evaluating the viability of higher radix design. Toward this end, this paper presents the design of a 16 x 16 position radix 4 simultaneous multiplier using the threshold I^2L logic described above.** The paper also establishes that multiplication by adding together copies of the multiplicand is more efficient than generating and summing single position partial products. The radix 4 design is compared to a computationally equivalent 32 x 32 bit binary multiplier in speed and circuit complexity (cost).

2. Multiplication

The process of multiplication of two four digit decimal numbers is shown in Fig. 1. Two digit partial products are formed for each position. The units position of these partial products are added

*This research was supported by the National Science Foundation under Grant No. ENG 76-09925.

**The multiplier is designed for LSI implementation, to which I^2L logic is very suited.

columnwise to form the product. The tens position (carries) are added with the next column.

The operations required to multiply two five position base R numbers are illustrated in Fig. 2. Except for the binary case when every $a_i b_j$ has only one position, each partial product $a_i b_j$, in general, is a two position number with a product P_{ij} and a carry position C_{ij} . The carry C_{ij} must be added with the next column as shown in Fig. 3.

Every column in Fig. 3 contains only single position numbers called summands. These are added columnwise to form the product. The number of summands S_i , in the column forming P_i , for a $n \times n$ position multiplier is given by:

$$S_i = 2(i-1) + 1 \quad 1 \leq i \leq n$$
$$S_i = 4n - 2(i-1) - 1 \quad n+1 \leq i \leq 2n$$

An $n \times n$ multiplier employing this scheme requires n^2 single position (1×1) multipliers to generate the partial products and adder circuits to sum the columns.

Multiplication can also be carried out by shifting and adding copies of the multiplicand. For the decimal example considered earlier, this is shown in Fig. 4.

Fig. 5 illustrates this scheme generalized to two five digit numbers in an arbitrary base R. The number of summands for a $n \times n$ position multiplier employing this scheme is given by:

$$S_i = i(R-1) \quad 1 \leq i \leq n$$
$$S_i = (2n-i)(R-1) \quad n+1 \leq i \leq 2n$$

A multiplier employing this scheme does not require single position multiplier circuits. Instead it requires circuits to gate replicas of the multiplicand into the summand adder inputs depending on the value of the controlling multiplier position.

In the future we shall refer to the first scheme as the multiplier adder scheme (MAS). The second scheme shall be referred to as the all adder scheme (AAS). Note that for the binary case ($R=2$) the two schemes reduce to one. This is the widely used scheme for parallel multiplication.

3. Summand Reduction

For both of the schemes discussed in the last section, there is no carry delay in the generation of the summands. Thus the summands are available after the single position multiplier delay time for MAS and the gating circuit delay time for AAS. As in binary multipliers, the important factor controlling speed is the summand addition time.

A number of schemes for fast addition of summands have been proposed for binary multipliers [4], [5]. Habibi and Wintz [6] compared these schemes. They reached the conclusion that the scheme due to Dadda [5] is the best from both speed and component cost considerations.

In Dadda's scheme the summand addition is carried out in two steps. In the first step, the original summand matrix is reduced to two numbers whose sum equals the product. The second step then adds these two numbers to generate the product. Step one is achieved without carry propagation delay by carrying out the summand matrix reduction in stages. First, the original matrix is reduced to another matrix with a smaller number of rows using full adders. (These reduce three inputs to two outputs.) The new reduced matrix is further reduced in the same way. This process is repeated until a two row matrix results. These two rows provide the two numbers to be added in step two.

Dadda also shows that for each stage of the matrix reduction, the full adders, (3,2) reducers, can be used optimally. Starting from the final two row matrix, we observe that this can be obtained from a three row matrix using (3,2) reducers. Thus the next matrix must have only 3 rows. A 3 row matrix can be split up into a 2 row matrix and a 1 row matrix. The 2 row matrix can be obtained from a 3 row matrix using (3,2) reducers. The next matrix should therefore have $3+1=4$ rows. Breaking up the 4 row matrix into two 2 row matrices which can each be obtained from a 3 row matrix we find that the next matrix should have 6 rows. Using this reasoning Dadda arrived at the following sequence for the number of rows in the matrix at each stage of the summand reduction

2, 3, 4, 6, 9, 13, 19

If the original matrix A had n rows, then the first transformation should reduce A to a matrix B, having a number of rows coinciding with the nearest term in the above sequence less than n . All the following matrices should have a number of rows coincident with the terms in the sequence in decreasing order till a two row matrix is reached. As an example the summand reduction for a 5×5 bit multiplier is shown in Fig. 6.

The scheme described above is not limited to binary multipliers. Dadda discussed the use of higher order counters for summand reduction. The same considerations can be used to apply the scheme to higher radix multiplication. In base 4, a 5 to 2 reduction is possible using a single adder. Going through the reasoning described above for base 4 and (5,2) reducers, the reduction sequence is found to be:

2, 5, 11, 26, 65

The stages in the summand reduction for a base 4 5×5 position multiplier are illustrated in Fig. 7.

The result of the summand reduction are two numbers which are added together to yield the final multiplication product. As for binary multipliers, this can be done in a carry ripple adder.

The next section presents circuits to implement base 4 AAS and MAS LSI multipliers using Dadda's scheme for fast summand addition.

4. Quaternary I^2L Circuits

Summand generation for the MAS requires single position quaternary multiplier circuits. The AAS requires gating circuits. Summand addition is done in both cases using Dadda's scheme. Therefore, both the AAS and MAS require (5,2) reducers for the summand reduction and a carry ripple adder for the final addition. An efficient LSI realization also uses 4,2; 3,2; and 2,2 reducers for the summand reduction so as not to leave some 5,2 reducer inputs unused. These circuits are next presented based on Signetics I^2L threshold logic described in [3].

While it is possible to realize any function using the complete set of logic primitives in [3], the resulting realizations are not necessarily efficient. It is well known that the class of linearly separable functions have efficient threshold logic implementation. This is taken advantage of in implementing the reducer and gating circuits.

5,2 Reducer

This circuit takes in five base four numbers and outputs the sum as a single two position number. The schematic is shown in Fig. 8. The five inputs are replicated and four copies of the sum are obtained. One copy is transformed from current sinking to current sourcing at the logic sum output X using a pnp mirror. The other three copies are compared, one each, to thresholds of 3.5, 7.5, and 11.5. If the sum is less than 3.5, all the threshold detectors remain in the on state and the sum is sent out at S. If $3.5 < \text{sum} < 7.5$, the threshold detector with level 3.5 turns off while the other two stay on. This results in four units of current being subtracted from the sum to yield the output at S. At the same time a carry value of 1 is summed with the zeros from the other two detectors and put out at C. The double complementing is required for the summing of the carries from the three detectors. In the same way for $7.5 < \text{sum} < 11.5$, $S = \text{Sum} - 4 \times 2$, and $C = 1 + 1 + 0 = 2$, and for $\text{sum} < 11.5$, $\text{Sum} - 4 \times 3$ can $C = 1 + 1 + 1 = 3$.

The other $k,2$ reducers needed ($2 \leq k < 5$) can be similarly constructed. The schematics are given in Fig. 9, 10, 11.

Gating Circuits

These circuits generate the summands for the AAS scheme. If X_i and Y_j are the i th and j th positions of the multiplicand (X) and multiplier (Y), then for each j , Y_j copies of X must be gated into the summand reducing adders. The schematic for gating circuits for a 16×16 position quaternary multiplier is shown in Fig. 12.

Four current sourcing copies of Y_j are obtained from a pnp current mirror. These are fed to 4 replication controllers, each of which controls the gating of 4 multiplicand positions into the adders. The replication controller gets 3 copies of each of the four multiplicand bits from 4 triplicating circuits. If Y_j if 0, the threshold detecting transistors of the replication controller sink all three copies of X_i , forcing all outputs to zero. If $Y_j = 1$, then the 0.5 threshold transistor turns off, outputting one copy of X_i . The other threshold detecting transistors similarly put out copies of X_i when their thresholds are exceeded. In this way, Y_j copies of X_i are gated into the summand reduction adders.

Single Position Multiplier

The product function is not linearly separable and thus does not have a simple threshold realization. Two approaches can be taken for implementing the 1×1 position multiplier required for summand generation in the MAS. The one favored by Dao [3] is multiplication by repeated addition. This makes the MAS very similar to the AAS because the single position multiplication is implemented using addition. But it is less efficient than the AAS because some of the 'summand' reduction is done separately as part of the single position multiplier. The AAS employs Dadda's scheme, which is known to be the most efficient scheme for summand reduction.

The other approach is to synthesize the single position multiplier using logic primitives. The primitives of the Su and Sarris algebra [7] are available in Signetics quaternary I²L logic. In this notation the expressions for the product and carry position of a single position multiplier are given below:

$$P = 1 \cdot (X^1Y^1 + X^3Y^3) + 2 \cdot (X^2Y^1 + X^1Y^2 + X^3Y^2 + X^2Y^3) + 3 \cdot (X^3Y^1 + X^1Y^3)$$

$$C = 1 \cdot (X \quad Y) + 2 \cdot (X \quad Y)$$

The single position multiplier can thus be implemented. This implementation is far more complex than Dao's implementation discussed earlier.

Carry Ripple Adder

This can be realized by cascading the (3,2) reducers (already described) in a manner directly analogous to the binary ripple adder.

Using the circuits described in this section, both the AAS and the MAS can be implemented. We next decide on a cost function to evaluate the complexity of these circuits before proceeding to

to compare them.

5. Cost Function

The high packing density of IL results because the two transistors that form the gate share some of the same semiconductor region and do not need isolation.

Fig. 13 shows a schematic for an I^2L gate. The base of the npn is common to the collector of the pnp while the base of the pnp transistor is common to the emitter of the npn. The emitter of the pnp, which is used as a current source, is common to all gates. Because this layout needs no component isolation, on silicon the entire gate takes up the space of a single multi-emitter transistor.

It is reasonable to take this structure as a unit for chip area cost in our circuit evaluations. Any transistor or other single component not part of this I²L gate structure and needing isolation is also assigned unit cost.

6. Comparing AAS With MAS

In this section, 16 x 16 position implementations of AAS and MAS are compared. The circuit cost is measured in terms of the unit of chip area selected in the last section. The delays are in terms of transistor switching delays.

The figures for the summand reduction stage for the two schemes are:

Summand Reduction	AAS	MAS
Cost	3095	2639
Delay	16	16

The cost of the gating circuits for the AAS is 784. Thus the total cost of summand generation and reduction for the AAS is 4860. For the MAS to be equally efficient, its cost for the summand generation must be less than $4860 - 2639 = 2221$. Summand generation in the MAS requires n^2 single position multipliers. For the MAS design to be competitive, each single position multiplier must, therefore, cost less than 9 units. Noting that cost of the single position multiplier implemented using the logic primitives in section 4 is an order of magnitude greater than 9, it is concluded that realizing an IL^2 threshold logic single position multiplier with a chip area less than 9 units is not feasible.

The gating circuits for the AAS add another 3 delays to the multiplication time. A single position multiplier built from logic primitives has 5 delays. Thus the MAS design does not have any speed advantage over the AAS.

It is therefore concluded that the all adder scheme of multiplication, achieved by adding copies of the multiplicand, is more efficient.

In the next section this design is compared with an I²L implementation of a computationally equivalent 32 x 32 bit binary multiplier.

7. Comparison of LSI Multipliers in Bases 2 and 4

With recent advances promising speeds comparable to TTL, I^2L technology is playing an important role in LSI design. It is therefore meaningful to chose this technology to compare a 16×16 position base 4 simultaneous multiplier with a 32×32 bit binary multiplier. The comparison is made in both speed and circuit cost.

Dao has shown [3] that a binary threshold full adder is more efficient than the conventional I^2L design for a full adder. Using threshold full adders and using Dadda's summand addition scheme, a 32×32 bit binary simultaneous multiplier can be built following a design analogous to the quaternary design described above. The cost and performance figures for the binary and quaternary realizations are listed below.

Summand	Base 2		Base 4	
	32 x 32 bits		16 x 16 bits	
	Cost	Delay	Cost	Delay
Generation	443	2	800	4
Reduction	6336	24	3905	16
Addition	192	96	176	48
Total	6971	122	4881	68

The figures indicate that the quaternary realization is faster and less expensive. There is a further substantial cost saving in the higher base design due to the fewer pins and bonding pads required.

The speed of the multiplier can be greatly improved by replacing the ripple carry adder by a fast look ahead adder. The design of such an adder in base 4 is very similar to the binary design because the carry only takes on the values 0 and 1. Base 4 full adders can be used which are no more complex than the binary circuits.

Conclusion

The design of a radix 4 simultaneous multiplier utilizing I^2L circuits, has been presented. Because of the relative efficiency of threshold logic implementations in I^2L to those using logic primitives, the multiplication was implemented by adding together copies of the multiplicand. This was shown to be more efficient than generating and summing single position partial products. The radix 4 design was also compared to an equivalent radix 2 multiplier (also mechanized with I^2L circuits), and was shown to be faster and less costly in terms of required chip area.

References

- [1] Dao, Tich T. "Electronics Newsletter," *Electronics*, McGraw-Hill, October 14, 1976, p. 26.
- [2] Dao, T. T., Russell, L. K., Preedy D. R., and McCluskey, E. J., "Multilevel I^2L with threshold gates," *ISSCC Tech. Dig.*, Feb. 1977, pp. 110-111.

- [3] Dao, Tich T., McCluskey, E. J., and Russell, L. K., "Multivalued integrated injection logic," *IEEE Trans. Comp.*, Vol. C-26, Dec. 1977, pp. 1233 - 1241.
- [4] Wallace, C. S., "A Suggestion for a Fast Multiplier," *IEEE Trans. Elect. Comp.*, Vol. EC-13, 1964, pp. 14-17.
- [5] Dadda, L., "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 36, March 1965, pp. 349-356.
- [6] Habibi, A. and Wintz, P. A., "Fast multipliers," *IEEE Trans. on Comp.*, Vol. C-19, No. 2, 1970, pp. 153-157.
- [7] Su, S. Y. H. and Sarris, A. A., "The relationship between multivalued switching algebra and Boolean algebra under different definitions of complement," *IEEE Trans. Comp.*, Vol. C-21, May 1972, pp. 479-485.

		2	5	4	9	
		1	7	3	2	
		4	10	8	18	
	6	15	12	27		
	14	35	28	63		
2	5	4	9			
4	4	1	4	8	6	8

Fig. 1

				a_5	a_4	a_3	a_2	a_1	
				b_5	b_4	b_3	b_2	b_1	
				a_5b_1	a_4b_1	a_3b_1	a_2b_1	a_1b_1	
			a_5b_2	a_4b_2	a_3b_2	a_2b_2	a_1b_2		
		a_5b_3	a_4b_3	a_3b_3	a_2b_3	a_1b_3			
	a_5b_4	a_4b_4	a_3b_4	a_2b_4	a_1b_4				
a_5b_5	a_4b_5	a_3b_5	a_2b_5	a_1b_5					
P_{10}	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1

Fig. 2

Fig. 3Fig. 4Fig. 5

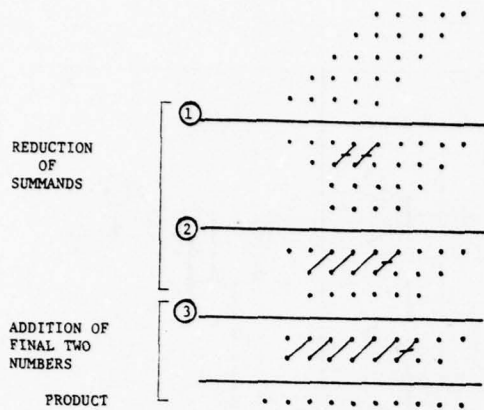


Fig. 6. 5 x 5 BINARY MULTIPLIER

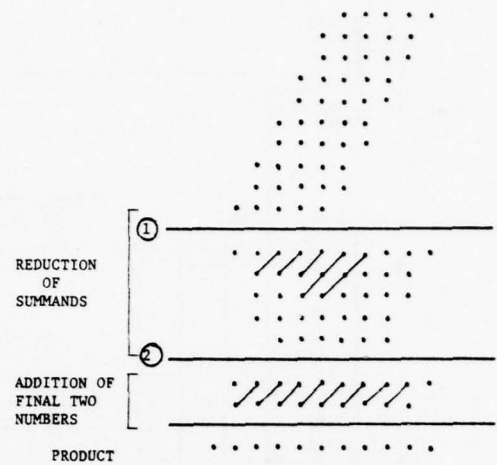


Fig. 7. 5 x 5 QUATERNARY MULTIPLIER (MAS)

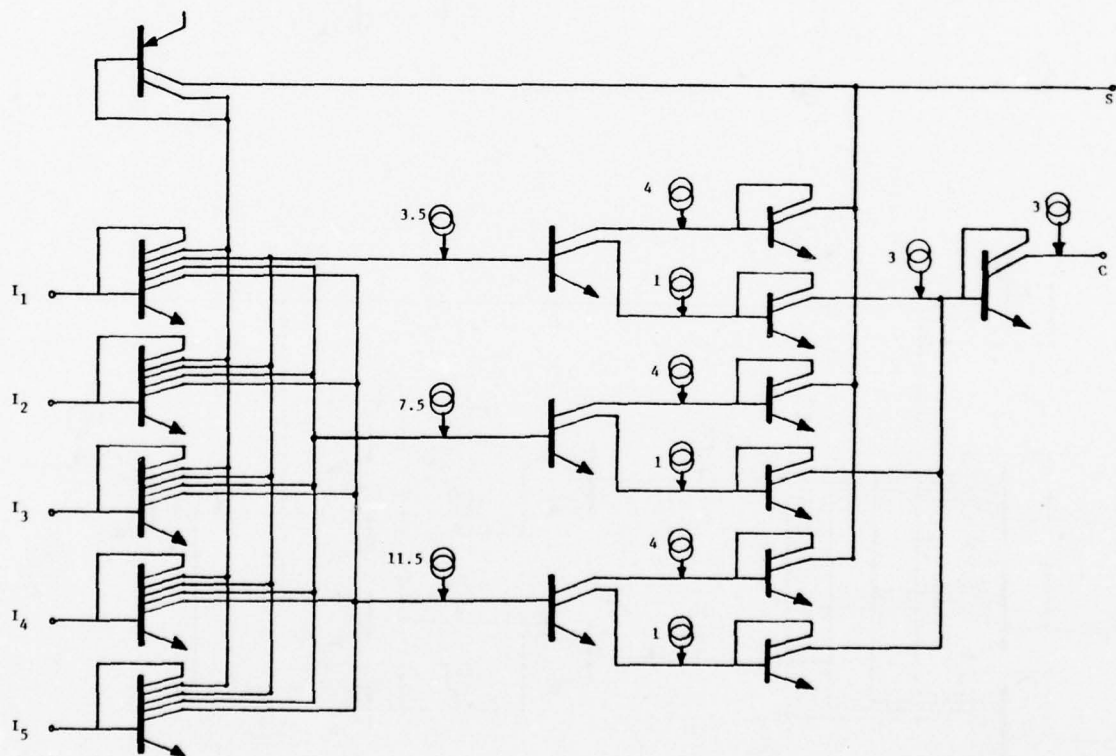


Fig. 8. (5,2) REDUCER

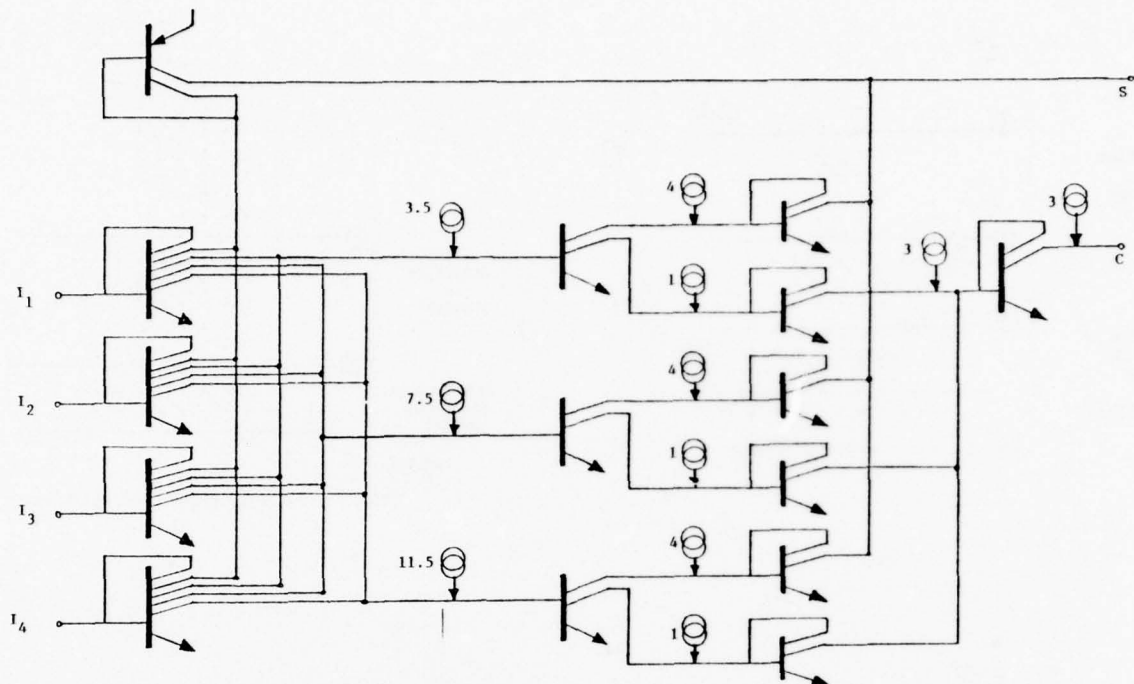


Fig. 9. (4,2) REDUCER

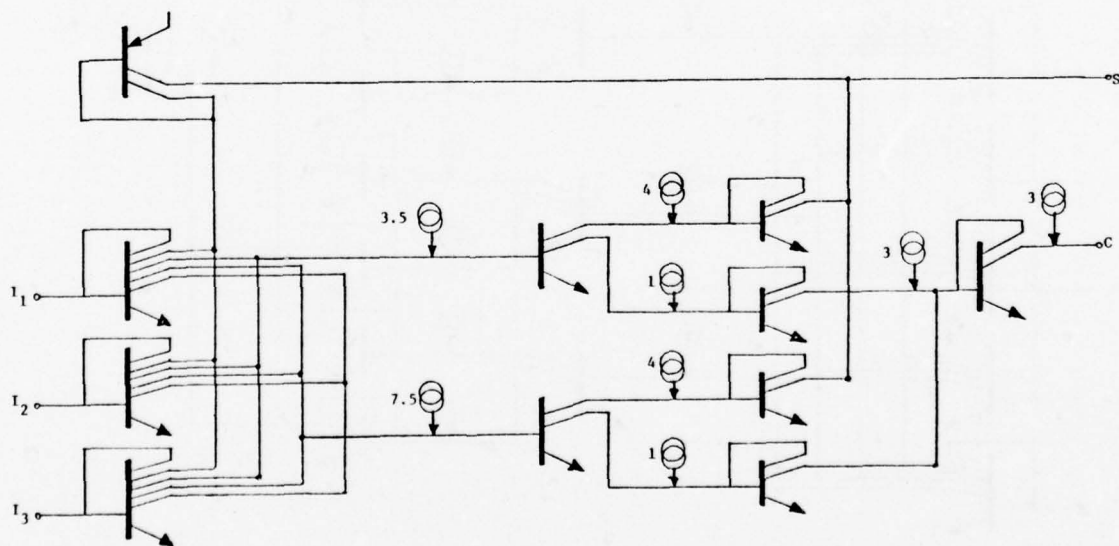


Fig. 10. (3,2) REDUCER

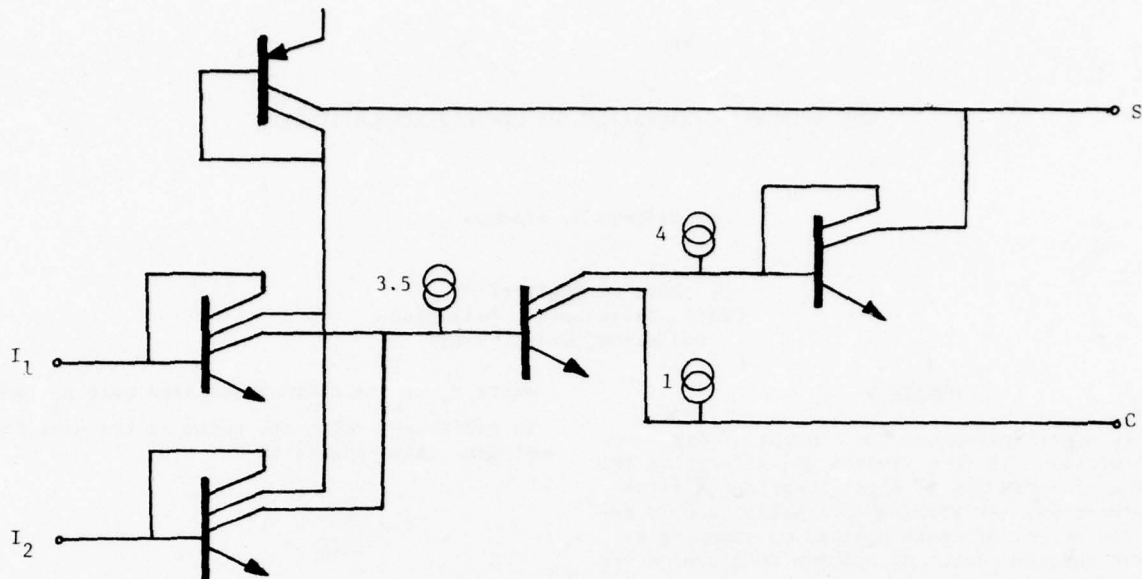


Fig. 11. (2,2) REDUCER

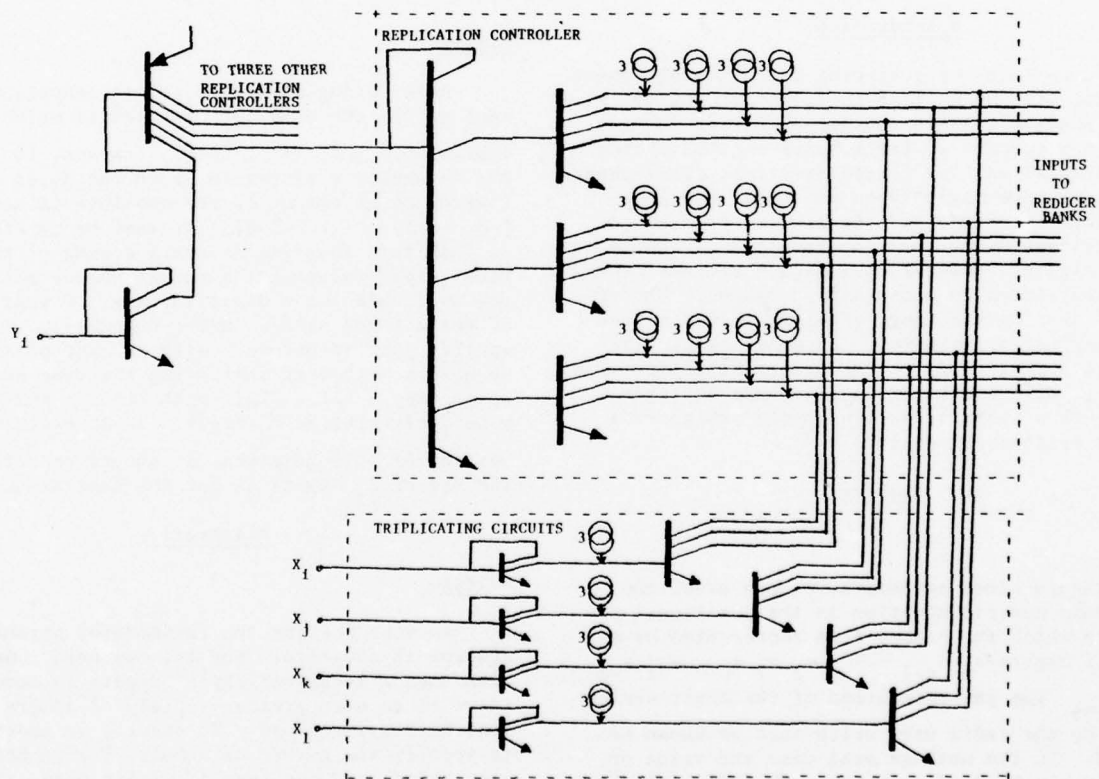


Fig. 12. AAS GATING CIRCUIT

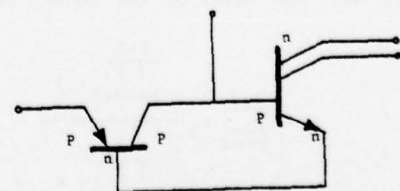


Fig. 13. I^2L GATE

THE ARITHMETIC PROPERTIES OF CERTAIN NUMBER SYSTEMS

Malcolm H. Steward

Div. of Engineering
Calif. State Univ., Fullerton
Fullerton, Calif. 92634

Abstract

This paper introduces the concept of digit range lowering. It then studies the effects of the selection of radix and of digit lowering on arithmetic processes. An attempt is finally made to indicate the effect of these choices on computer arithmetic section cost. An opinion is given on the optimum choice.

Introduction

When we think of designing computing equipment to operate with unfamiliar radices, it pays to return to fundamentals to reexamine numbers and arithmetic processes to lay a basis for the choice of fundamental machine characteristics. This paper will survey some digit types and their use in arithmetic. The study made will be restricted to number systems using positive radices and non-redundant digits. However, the digits are not necessarily restricted to non-negative values. The effect of lowering the range of digit values is one of the key factors studied. In order to provide designers with a guide to the proper selection of a number system, an attempt is then made to foresee the effects of this selection on the design of a computer arithmetic section.

Numbers

Radices

Although other systems have been used, the most common numeric notation is the positional notation in which the number A is represented by a string of digits $a_n a_{n-1} \dots a_2 a_1 a_0 \dots a_{-1} a_{-2} \dots a_{-m}$. The interpretation of the digit string depends on the radix used which must be known beforehand. In the most general case the value of the radix may be different for each different digit position. The value of the number is then

$$A = \sum_{i=-m}^n a_i \frac{\prod_{j=-m-1}^{i-1} r_j}{\prod_{k=-m-1}^{-1} r_k} \quad (1)$$

where r_i is the radix associated with a_i , and r_{-m-1} is arbitrary. When the radix is the same for all digits, this reduces to

$$A = \sum_{i=-m}^n a_i r^i \quad (2)$$

Although negative values may be used for the radix r , only positive integral values will be considered here.

Digits

Restricting ourselves to non-redundant digits, each a_i has one of a set of r values which are generally $\{0, 1, 2, \dots, r-1\}$. However, it is possible to employ a system in which the digit range is lowered by an amount q , the possible values being $\{-q, -q+1, \dots, r-1-q\}$. It must be carefully noted that this lowering is not a coding of the previous digit values but a change in the set of values available for a digit to have. 0 must be one of the allowed values, hence $0 \leq q \leq r-1$. Eq. (2) applies just as before. With so many possibilities we need a method of indicating the type of digit being used. Let a digit with radix r and lowering q be designated an r_q digit. As an exercise in the meaning of this notation, it should be noted that the use of 2_1 digits is not the same as using $r=-2$.

Arithmetic

Adders

In most designs the fundamental arithmetic operation is addition. For this we need adders, but with such a large variety of digits to consider, there is an even greater variety of adders that must be distinguished. To specify an adder we need to specify the number of inputs, the number of columns added, and the type of digits used. If all the inputs to an adder consist of only one digit each, it will be called a digit adder. This is in contrast with number adders whose inputs are multi-digit numbers. Let a $j, (r_q)_1, (r_q)_2, h$ adder be an adder with j inputs of one $(r_q)_1$ digit each plus one further input of an $(r_q)_2$ digit (this may be a carry digit) plus a constant input of value h . Its outputs are an $(r_q)_1$ sum digit and a carry. If

there is no constant input, h may be omitted. If all the variable digits are of the same type, $(r_q)_2$ may be omitted, and j will indicate the total quantity of variable input digits. With these definitions, binary half and full adders are designated $2,2_0$ and $3,2_0$ adders respectively.

An adder for numbers of k digits each (a k -column adder) will be designated a $j,k,(r_q)_1,(r_q)_2,h$ adder. This adder has j inputs of $k(r_q)_1$ digits each plus an $(r_q)_2$ carry from column to column and a constant input of value h in each column. An adder for all the columns of the numbers in use (k not specified) will be designated a $jN(r_q)_1,(r_q)_2,h$ adder. h and/or $(r_q)_2$ may be omitted for number adders just as for digit adders. As a second exercise in the use of this notation it may be noted that one $2,2_0$ adder and seven $3,2_0$ adders may be cascaded in a row to form a $2,8,2_0$ adder. These details may be spelled out thus: $2,2_0; 7 \times 3,2_0 = 2,8,2_0$.

Subtraction, complements, and negatives

When adders are used, subtraction is accomplished by adding the negative of the subtrahend. This requires having a means of representing negative numbers and a means of obtaining the negative of an operand. The use of complements for this is well known in the 2_0 and 10_0 cases. To generalize on this idea let us define the complement of a digit as

$$\bar{a}_i = r_i - 1 - 2q_i - a_i \quad (3)$$

Then the complement of a number is \bar{A} derived from A by replacing each a_i by \bar{a}_i . This definition allows the complement to be expressed with digits of the same type as the number it is derived from. It can thus be used as an input to the same type of adder. However, a correction may be required when this type of complement is used.

When $q = 0$, only positive values can result from eq. (2). In order to distinguish between numbers and complements a sign must be affixed to the digit string. This generally takes the form of another digit, with $0 =$ positive, placed at the most significant end of the string. When this is done we will assume n to have been augmented by 1 so that eqs. (1) through (3) continue to hold. The sign digit can be of the same type as the others, or it can be a 2_0 or any other type of digit. When $q = r-1$, only negative values can result from eq. (2), and the problems are similar to those with $q = 0$. If $0 < q < r-1$, any finite value, positive or negative, with a tolerance of $\pm \frac{1}{2} r^{-m}$, can be put in the form of eq. (2) without need for a sign digit. However, in most cases the calculation of $-A$ from A is not simple and will involve the possibility of borrows that can propagate as far as the entire length of the number. The corrections required when using the complement are less onerous.

When the complement is used, most digit types require a sign to distinguish the complement.

In general

$$\bar{A} = \sum_{i=-m}^n (r - 1 - 2q - a_i) r^i \quad (4)$$

$$= \sum_{i=-m}^n (r - 1 - 2q) r^i - A \quad (5)$$

$$A - B = A + \bar{B} - \sum_{i=-m}^n (r - 1 - 2q) r^i \quad (6)$$

$$= A + \bar{B} - r^{n+1} + r^{-m} + 2q \sum_{i=-m}^n r^i \quad (7)$$

Hence subtraction may be accomplished by adding the complement and making a fixed correction. If the word length is limited and overflow prohibited or otherwise taken care of, r^{n+1} may be ignored along with carries out of the n^{th} column. There remains an additive correction of $2q$ in every column plus a 1 in the least significant position. This is the standard procedure when $q = 0$. $2q$ will be an allowed digit value if

$$2q \leq r - 1 - q \quad (8)$$

$$\text{or if } q \leq \frac{r-1}{3} \quad (9)$$

An alternative is to use eq. (6) and add $2q - r + 1$ in every column. This will be an allowed digit value if

$$-q \leq 2q - r + 1 \leq r - 1 - q \quad (10)$$

$$\text{or if } \frac{r-1}{3} \leq q \leq 2 \frac{r-1}{3} \quad (11)$$

A particularly simple case is available when r is odd. In this case we may choose $q = (r-1)/2$ with the result that the correction vanishes and $\bar{A} \equiv -A$. No sign digit is required for this case. With this selection, $q = r-1-q$, and the range of allowed digit values is symmetrical about 0. In recognition of this fact an $r \frac{r-1}{2}$ digit may also be designated an r_s digit.

The permissible number of adder inputs

To my knowledge adder designers have universally chosen to restrict the carry to a single digit. This in turn restricts the number of inputs that the adder can handle. With the carry limited to a single digit, the output of a j,r adder consists of an r_q carry digit K and an r_q sum digit S and has the value $Kr + S$. This must lie in the range

$$(-q)r - q \leq Kr + S \leq (r-1-q)r + (r-1-q) \quad (12)$$

$$\text{or } -q(r+1) \leq Kr + S \leq (r-1-q)(r+1) \quad (13)$$

Hence the limit on j is

$$j \leq r + 1 \quad (14)$$

independent of q . For a jN_r adder the limit is

$$j \leq r \quad (15)$$

since one input in all but the least significant column must be reserved for the carry.

The effect of q on adders

Having noted that q does not affect the number of inputs that an adder may accept, we may well ask what effect, if any, q has on adder design. Let a j, r_0 adder be designed using the (voltage or current) levels L_1, L_2, \dots, L_r to represent the digit values $0, 1, \dots, r-1$. If its inputs are a_u , then its logic will be such that

$$K'r + S = \sum_{u=1}^j a_u \quad (16)$$

Now let the levels L_1, \dots, L_r be reinterpreted to represent the digit values $-q, -q+1, \dots, r-1-q$. If inputs b_v are now applied to the same adder as above, its action will be

$$(K' + q)r + S' + q = \sum_{v=1}^j (b_v + q) \quad (17)$$

$$\text{or } K'r + S' = \sum_{v=1}^j b_v - q(r + 1 - j) \quad (18)$$

If $j = r + 1$, the term on the right vanishes and we see that the same identical circuit serves as an $r+1, r_0$ adder regardless of q . If $j < r + 1$, the circuit will serve as a j, r_0 adder provided the correction $q(r + 1 - j)$ is made. In other words a $j, r_0, q(r+1-j)$ adder serves as a j, r_0 adder. Conversely a $j, r_0, -q(r+1-j)$ adder will serve as a j, r_0 adder.

The foregoing conclusions may also be understood from another point of view as follows: If not all the $r+1$ possible inputs to an adder are used, the unused inputs should be held at the level representing the value 0. However, this level is different for different values of q . If, for instance, an r_0 adder is to be used to add r_q digits, any unused input must be connected to the level that represents 0 in the r_q system. This is the level that was designated q in the design of the adder. On the other hand, if an r_q adder is to be used to add r_0 digits, any unused input must be connected to the level that represents 0 in the r_0 system. This is the level that has been designated $-q$ in the design of this adder.

Multiplication

The multiplication of two numbers consists of the multiplication of all the combinations of a digit of the multiplicand by a digit of the multi-

plier (digit products), followed by the summation of the digit products arrayed in their proper positions. Fig. 1 shows the case of equal numbers of digits in the two factors. This paper will consider only this case; however, the extension to the case of unequal numbers of digits is immediate. The radix points dividing the integral and fractional parts of the operands and the product are not shown since they have no effect on the design.

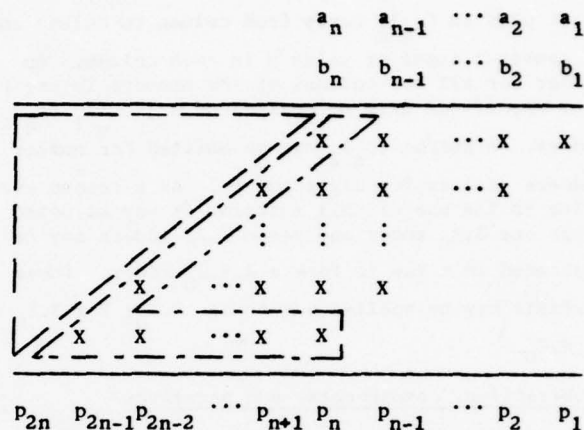


Fig. 1. The array of digit products.

The X's in the figure indicate digit products. In the cases $2_0, 2_1$ and 3_s , in which the greatest absolute value of a digit is 1, the digit products are limited to a single digit. In all other cases the digit products must be considered to consist of a digit at the location of the X plus a carry into the succeeding columns. If the complexity of design is to be held down, the carry should be limited to not more than one digit. This requires the satisfaction of 3 conditions, to wit: the square of the most positive allowed digit value, the square of the most negative allowed digit value, and the product of the most positive and the most negative allowed digit values must all be expressible with no more than two digits. These conditions may be expressed as

$$(r - 1 - q)^2 \leq (r - 1 - q)(r + 1) \quad (19)$$

$$q^2 \leq (r - 1 - q)(r + 1) \quad (20)$$

$$-q(r - 1 - q) \geq -q(r + 1) \quad (21)$$

Eqs. (19) and (21) hold for all values of q . For eq. (20) to be valid, however, we must have

$$q \leq -\frac{r+1}{2} + \sqrt{\frac{(r+1)^2}{4} + r^2 - 1} \quad (22)$$

From eq. (22) we have

r	2	3	4	5	6	7	8	9	10	11	12
q_{\max}	0	1	2	2	3	4	4	5	5	6	7

It may be seen that $q_{\max} \geq \frac{r-1}{2}$, except if $r = 2$, and all r_s digits meet the stated condition.

When sign digits are used, and corrections are made in accordance with eq. (7), the algorithm used may involve filling the dashed triangle and/or special treatment of the contents of the dot-dashed area. When only positive numbers are used, or if the corrections are made according to eq. (6), no special treatment is required. The use of r_s digits is the outstanding example of this.

The array of digit products may be summed in one step in an array multiplier, or an add-shift algorithm may be used to add a limited number of rows at a time. Using an rNr adder, $r-1$ rows may be added to a previous partial product in each step. An independent addend may be included in the first step for uniformity (thus calculating $A B + C$), or zero may be inserted, or the first step may sum r rows. The vertical position of any digit product is of no consequence, hence we may reshape the array into an isosceles triangle as shown in fig. 2. This shape is more convenient for thinking about the design of an array multiplier. The area

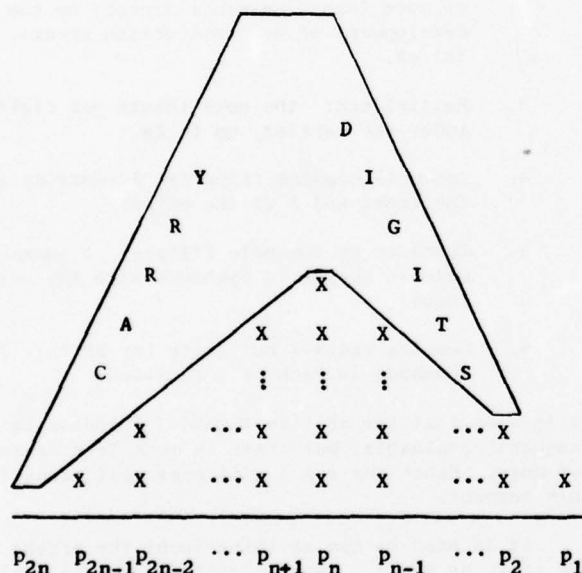


Fig. 2. The multiplier array reshaped.

marked Carry Digits reminds us that the digits to be summed in any column include multiplication carries and adder carries from the previous column as well as the indicated product digits.

Division

Division algorithms are put together from add-subtract operations and shifts. A key to building the divider is the ability to detect the sign of the remainder after each step. Sign detection is also required for some conditional operations in

the computer. The sign is explicit in the sign digit when one is used. When there is no sign, circuits must be available to determine the sign of a number. The sign-detector circuit must use all the digits as inputs. This is obviously more complicated than referring to a single digit, but fortunately relatively simple circuits (per digit) are available.

Roundoff

Computing algorithms frequently include the roundoff of intermediate or final results, particularly following multiplication. A great deal of study has been made of the question of what degree of roundoff is permissible for various particular computations, and of whether truncation is an acceptable alternative. It behooves us, therefore to inquire into how roundoff may be accomplished in various number systems.

Suppose the number A is to be rounded off such that a_m is the last digit retained. Let A be divided into

$$A = A' + A'' = \sum_{i=n}^{-m} a_i r^i + \sum_{i=-m-1}^{-\infty} a_i r^i \quad (23)$$

Then A'' is discarded and a correction $c r^{-m}$ is added to A' , where c is a 3_s digit. The problem is to determine c . Before proceeding with the problem we note that for finite strings of equal digits

$$|A''| < \sum_{i=-m-1}^{-\infty} |a| r^i = \frac{|a|}{r-1} r^{-m} \quad (24)$$

Now, $c = 1$ if and only if $A'' \geq \frac{1}{2} r^{-m}$. If r is even, $A'' \geq \frac{1}{2} r^{-m}$ if

$$a_{-m-1} a_{-m-2} a_{-m-3} \dots \geq \frac{r}{2} 0 0 \dots \quad (25)$$

If r is odd, $A'' \geq \frac{1}{2} r^{-m}$ if

$$a_{-m-1} a_{-m-2} a_{-m-3} \dots > \frac{r-1}{2} \frac{r-1}{2} \frac{r-1}{2} \dots \quad (26)$$

$c = -1$ if and only if $A'' \leq -\frac{1}{2} r^{-m}$. If r is even, $A'' \leq -\frac{1}{2} r^{-m}$ if

$$a_{-m-1} a_{-m-2} a_{-m-3} \dots \leq -\frac{r}{2} 0 0 \dots \quad (27)$$

If r is odd, $A'' \leq -\frac{1}{2} r^{-m}$ if

$$a_{-m-1} a_{-m-2} a_{-m-3} \dots < -\frac{r-1}{2} - \frac{r-1}{2} - \frac{r-1}{2} \dots \quad (28)$$

It should be noted that there is no single value of q that permits both positive and negative c since the digit values required to occasion at least one or the other will be outside the allowed set.

The above results can be divided into three cases. If r is even and $q = 0$, c can be determined from a_{-m-1} only. If r is even and $q \neq 0$, or if r is odd and $q \neq \frac{r-1}{2}$, perfectly proper roundoff can

be performed only by using a test that must be extended to as many as all the discarded digits, depending on the values encountered. An algorithm based on testing one, two, or any number less than all the discarded digits will have a corresponding amount of bias error. If r_s digits are used, $c \approx 0$, and truncation is identical to proper roundoff. Furthermore, repeated truncation always gives the same result as one large truncation, in contrast to what may happen if numbers using other types of digit are repeatedly rounded. The error caused by rounding the decimal number 0.444445 one place at a time may be compared with the freedom from such problems of r_s numbers.

The choice of a number system

The choice of radix pervades the entire machine design, since it not only affects circuit design, but it also affects word length and determines the maximum number of operands that may be added conveniently in one step.

Word length

If a word is to contain I bits of information, the number of digits required is

$$w = I / \log_2 r = I \frac{\log 2}{\log r} \quad (29)$$

assuming that all numerical values expressible with the given number of digits are equally probable. This equation gives the following values which are tabulated here for reference:

r	w/I	I/w	r	w/I	I/w
2	1	1	7	0.356	2.807
3	0.631	1.585	8	0.333	3.000
4	0.500	2.000	9	0.315	3.170
5	0.431	2.322	10	0.301	3.322
6	0.387	2.585			

The advantage of larger values of r tapers off as r is increased. At the same time circuit complexity grows, and the required tolerances become tighter and tighter.

The optimum radix

If cost per digit of w were known as a function of r , an optimum r for minimum cost could be found. The assumption has been made numerous times that cost per digit is proportional to r . This leads to the result that the optimum radix is 3. This assumption is probably reasonable in a long distance communication system using a modulation system for which the bandwidth is nearly proportional to r for a given number of digits per second.

A different assumption is that on a printed circuit board the integrated circuit chip is only a minor part of the cost, so that total cost will vary monotonically with the number of pins on the I.C.'s, and thus with w . This would call for the use of the maximum possible radix. In this case the choice of r is limited only by the circuits a-

available, which are limited in principle only by the ability to hold close tolerances in production.

Current reality is that the cost of circuitry is not yet trivial, as witness the prices of commercially available array multipliers. Hence the optimum radix is not yet very large since the growth of circuit complexity with r offsets the advantage in pin reduction.

Other considerations

The special case of $r = 10$ has obvious advantages when human-readable input-output is required, provided practical circuits are available. Of course, this case has no special importance when human readability is not required.

Setting the questions of pins and input-output aside, an important advantage of a larger r lies in the ability to add more summands at a time, provided there are that many summands to be added. In surveying requirements we see the following cases:

1. Address modification: stored address + base address + index--3 summands.
2. Accumulators: contents + data--the use of more than 2 summands depends on the development of new instruction repertoires.
3. Multipliers: the more inputs per digit adder the merrier, up to $2w$.
4. Two-pole digital filters: 3 summands at the input and 3 at the output.
5. Cascades of two-pole filters: 5 summands when an output is combined with the next input.
6. Complex radix-2 butterfly for FFT's: 3 summands in each of 4 equations.

It is seen that the ability to add 3 summands is frequently valuable, but there is much less demand for more. Hence any $r \geq 3$ will meet most needs in this respect.

It is hard to say anything about the effect of the radix on speed. Shorter words generally imply greater speeds if the delay per digit is constant. However, larger radices imply more complex circuits so that the delay per digit will probably increase.

Circuit cost

The arguments given above are all well and good, but we need some actual cost figures before we can decide anything. An idea of relative circuit cost for adders can be had from the component counts of various adders using LHT gates*. In making these counts a transistor and an associated

* LHT gates are one of several types invented by the author. A planned paper on the subject has not yet appeared.

Schottky diode are counted as one component. The results are:

Adder type	Components/digit			Components/digit x w/I		
	No. of summands			No. of summands		
	2	3	5	2	3	5
2N2 ₀	9	2x9	4x9	9	18	36
3N3 _s	16	16	2x16	9.9	9.9	19.8
3N4 _{0,3} ₀	20	20	2x20	10	10	20
3N5 _{s,3} _s	23	23	2x23	10.1	10.1	20.2
5N5 _s	30	30	30	12.9	12.9	12.9
2N10 _{0,2} ₀	28	2x28	4x28	8.4	16.9	33.7

The decimal adder referred to in the table is, in reality, made up of alternating binary and quinary stages. Hence 2 terminals are needed for each digit, and the pin reduction is only slightly better than for ternary digits. From the table we see that the 3_s circuits are optimum on the basis of component count if 3 summands are to be added. The 3N4_{0,3}₀ and 3N5_{s,3}_s adders come in close behind. The 5N5_s adder might be good for some applications, but the tolerance requirement is much tighter here than for the other cases.

Summands are most numerous in an array multiplier. The effect on multiplier design of increasing r may be seen by comparing multipliers for w = 16 bits, 10 trits (ternary digits), 8 quits (quaternary digits), and 7 quints (quinary digits). Words of these lengths hold very nearly the same amount of information. With the help of fig. 2 we find that the numbers of digit multipliers and of digit adders needed for these multipliers are

r	2	3 _s	3 ₀	4	5
digit mults	256	100	100	64	49
digit adders	240	50	99	45	27

provided the adders can accept r+1 inputs in each case. Otherwise more are required. The increase in adder requirement seen from 3_s to 3₀ is due to the appearance of two-digit digit products at this point. The apparent advantage of the quaternary and quinary numbers is more than offset by the complexity of the individual circuits needed. Thus, once more, the 3_s system uses the fewest components to process an equivalent amount of information.

Conclusions

The effects of digit range lowering and the limitations imposed by the radix used have been studied. It has been seen that the radix chosen places a limit on the number of addends that may be conveniently summed at once. It has also been seen that odd radices are to be preferred because of the advantages of symmetric digits. The latter simplify addition, negation, multiplication, and round-off. It has been found that adders designed with

or without digit range lowering may be used to add inputs with any desired lowering provided (at worst) a suitable constant input is included.

In discussing cost it was noted that increasing the radix reduces the number of integrated circuit pins and usually the number of individual circuits while increasing the complexity of the individual circuits. The 3_s system uses the fewest components both in adders for typical requirements and in multipliers. It may well be the optimum. When I.C. pin count is considered, the optimum radix may possibly increase to 5. If this choice is made, the 3N5_{s,3}_s adder will meet most requirements. The ability of either of the recommended systems to add 3 addends at a time will lead to interesting new designs. More than that will seldom be useful.

DIGITAL CALCULUS

Samuel C. Lee, Senior Member, IEEE
Yousef M. Ajabnoor, Student Member, IEEE

School of Electrical Engineering and Computing Sciences
University of Oklahoma
Norman, Oklahoma 73019

ABSTRACT

This paper attempts to establish a mathematical foundation for the variational analysis of multivalued switching systems. The mathematics in question is termed digital calculus, which includes Boolean calculus and binary vector Boolean calculus as special cases.

In this paper the concept of Boolean difference (Boolean derivative) has been extended to multivalued (m -valued) switching algebra, where m need not be an integer power of 2. The derivatives of multivalued switching functions of potentially implementable multivalued algebras and their properties are presented. It is shown that every multivalued switching function of these algebras has a MacLaurin series expansion and a Taylor series expansion. As an example of the applications of digital calculus to multivalued switching systems design, the derivation of tests for fault detection of stuck-type faults in multivalued combinational circuits is presented and illustrated by an example.

1. INTRODUCTION

The concept of Boolean derivative (Boolean difference) of a Boolean function, introduced by Reed [1], has been more thoroughly investigated in two papers due to Akers [2] and to Sellers, Hsiao, and Bearson [3]. In both these papers, various differential operators are introduced and described in connection with their application to switching problems. Boolean differential calculus, introduced by Thayse in 1971 [4] encompasses and generalizes the algebraic concepts introduced by the former researchers. Recently, Lee [5, 6, Reference 7, chapter 2 and 3] further extended it to binary-vector Boolean differential calculus. It has been shown [6] that any Boolean function can be analytically represented by a binary-vector switching function. Properties and canonical forms of it were presented.

Recently, there is a growing interest in research in multivalued switching systems. This is mainly due to the following reasons:

(1) The size and speed of binary circuit components and memory devices approaching to its limit. The binary number system has been used throughout the entire development of computer technology. The growth of

computation systems and the need to process increasing volumes of data faster has resulted in the development of large-scale integration (LSI) integrated circuits. However, the volume of data continues to increase while the circuit components and memory devices approach to their practical limit in size and speed. The design of computation systems in other number systems seems to be a logical solution to the continued increase in volume of data to be processed by digital computers.

(2) Pin limitation problem of binary LSI. One of the most important problems in designing very large binary LSI is the pin limitation of the integrated circuits. Multivalued switching circuits allow each input pin to accept and each output pin to deliver more information; therefore, for the same amount of information transfer, the total number of pins required in an integrated circuit chip containing multivalued switching elements is much less than that of an integrated circuit chip containing multivalued switching elements is much less than that of an integrated circuit chip with binary elements. As a simple example, in binary system, eight input pins of an integrated circuit chip are sufficient to accept any of the 256 numbers whereas only six input pins are needed (a one fourth pin number reduction) if an equivalent ternary switching circuit is used.

(3) The need of decimal-to-binary and binary-to-decimal conversions. Since man works with decimal numbers, it is desirable to have a computer which accepts decimal numbers, processes them, and produces decimal numbers directly to eliminate the processes of converting back and forth between decimal and binary numbers which are required in binary-numbers which are required in binary-number based computers.

(4) Increasing reliability and reducing cost of analog and digital circuitry. In the past, few multiple-valued switching circuits have been built because of the high cost and low reliability. However, the price for digital circuits has been decreasing and will continue to drop in the future. At the same time, the industry is building more reliable and testable multivalued switching circuits.

The purpose of this paper is to extend Boolean differential calculus [4] binary-vector Boolean calculus [6] to a more general calculus which will be applicable to not only binary and binary-vector switching systems, but also to multivalued switching systems (such as 3-valued and general m -valued switching systems, where m need

not be an integer power of 2), which are potentially implementable for practical applications. In Section II, two potentially implementable multivalued algebras and explicit relationships between them are presented. The partial derivative of a multivalued switching function defined on these algebras and its properties are introduced in Section III. Based on this definition of the partial derivative, the MacLaurin and Taylor series expansions of a multivalued switching function are found and presented in Section IV. Finally, the application of the partial derivative to the derivation of fault-detection tests for multivalued combinational circuits is described in Section V.

II. TWO POTENTIALLY IMPLEMENTABLE MULTIVALUED ALGEBRAS

The primary objective of much of the recent work in multivalued switching systems is to devise a set of practically implementable basic functions, and to develop an algebra such that functions of arbitrary complexity may be represented in terms of simple algebraic combinations of the basic functions. If in addition, the choice of basic functions and the algebra permits the development of a technique to simplify in some useful sense the complexity of the functional representations developed, then these representations are of considerable additional interest. Finally, it is potentially advantageous to devise a system which is adaptable to any switching function, regardless of the choice of integral base of the switching variables involved.

DEFINITION 1 A multivalued algebra is said to be potentially implementable for practical applications, or simply, potentially implementable if it possesses the following three properties:

- A set of practically implementable basic functions which constitute a functionally complete set for realizing any switching function defined on the algebra.
- Canonical forms.
- Well-defined function minimization techniques.

Among the multivalued switching algebras published in the literature, [15, 16, 26] it is found that two algebras, one due to Vranesic, Lee, and Smith [8] and the other due to Su and Sarris [9] are potentially implementable. For convenience, they will be referred to as Algebra A and Algebra B, respectively.

Algebra A

This algebra is defined as follows:

- It contains a set of variables (x, y, z, \dots) which can assume m logic values from the set $Q = \{0, 1, \dots, m-1\}$, $0 < 1 < \dots < m-1$.
- There exists an equivalence ($=$) operation, that is

$x = x$
if $x = y$, then $y = x$,
if $x = y$ and $y = z$, then $x = z$.

- It has $2m + 2$ basic operations (basic set).

- Two-Element Operations

$$x + y = \max(x, y)$$

$$\text{product } x y = \min(x, y)$$

where $\max(x, y)$ and $\min(x, y)$ indicate the highest and the lowest values of (x, y) , respectively.

- m Unary "Inverter" Operations:

$$x^K = K \text{ iff } x = 0$$

$$= 0 \text{ otherwise}$$

for $K \in Q$.

- m Unary "Clockwise Cycling" Operations:

$$x = (x + M) \bmod m$$

where $M \in Q$.

- The two-element operations obey the idempotent, commutative, associative, distributive, and absorption laws.

Post 10 showed that the cycling operation and the product operation are a functionally complete set, and so this expanded collection of operations is also functionally complete. Any n -variable m -valued switching function $f(x_1, \dots, x_n)$ has a sum-of-products-of-sums canonical

form:

$$f(x_1, \dots, x_n) = \sum_{k=1}^{m-1} \left\{ \prod_{V | f(V)=K} \left(\sum_{i=1}^n \overset{m-v_i}{\rightarrow} x_i \right) \right\}^K \quad (1)$$

where $V | f(V) = K$ is the set of vertices $V = v_1, \dots, v_n$ for which $f(V) = K$.

Example 1: The sum-of-products canonical form of the multivalued switching function described by the truth table of Table 1 is

$$f(x, y) = \left[(x + y) \left(\frac{1}{x + y} \right) \right]^1 + \left[(x + \frac{1}{y}) \left(\frac{1}{x + \frac{1}{y}} \right) \right]^2 \quad (2)$$

Example 2: The canonical form of the 5-valued switching function of Table 2 is

$$f(x,y) = \left[\begin{matrix} 1 & 1 & 4 & 1 & 3 & 1 & 2 & 1 & 1 \\ (\bar{x} + y) & (\bar{x} + \bar{y}) & (\bar{x} + \bar{y}) & (\bar{x} + \bar{y}) & (\bar{x} + \bar{y}) & (\bar{x} + \bar{y}) & (\bar{x} + \bar{y}) & (\bar{x} + \bar{y}) & (\bar{x} + \bar{y}) \end{matrix} \right]^1 + \left[\begin{matrix} 3 & 2 & 1 & 4 & 3 \\ (x + \bar{y}) & (x + \bar{y}) & (x + \bar{y}) & (x + \bar{y}) & (x + \bar{y}) \end{matrix} \right]^2 + \left[\begin{matrix} 4 & 2 \\ (x + y) & (\bar{x} + \bar{y}) \end{matrix} \right]^3 \quad (3)$$

A number of important properties exhibited by the selected basic set provide a means for algebraic manipulation.

THEOREM 1

$$(a) (x_1 + x_2 + \dots + x_n)^K = x_1^K + x_2^K + \dots + x_n^K$$

$$(b) (x_1 \cdot x_2 \cdot \dots \cdot x_n)^K = x_1^K \cdot x_2^K \cdot \dots \cdot x_n^K$$

where $1 \leq K \leq m-1$.

THEOREM 2

$$(a) (x \cdot x^K) = 0$$

$$(b) (x \cdot x^K)^M = M$$

$$(c) x \cdot \bar{x} \cdot \dots \cdot \bar{x}^{m-1} = 0$$

where $1 \leq K \leq m-1$ and $1 \leq M \leq m-1$.

Many other properties of this algebra were given by Vranesic et al. [8].

Example 3: Applying Theorem 2, the function of Eq. (2) may be expressed as

$$f(x,y) = \left[(x + y)(\bar{x} + \bar{y}) \right]^1 + \left[(x + y)(\bar{x} + \bar{y}) \right]^2$$

$$= (x + y)^1 + (\bar{x} + \bar{y})^1 + (x + \bar{y})^2 + (\bar{x} + y)^2$$

$$= x^1 y^1 + \bar{x}^1 \bar{y}^1 + x^2 \bar{y}^2 + \bar{x}^2 y^2 \quad (4)$$

A procedure for minimizing any function in canonical form of Eq. (1) has been found. A computer program for implementing this procedure was written and used for synthesis of multivalued switching function of this

algebra [8].

Operations of the above set are a natural choice because of their potentially simple implementation. The sum, product, and inverter operations can be implemented using ordinary electronic gates with minor modifications; the cycling operation can be implemented by a universal cycling gate which has been designed by Vranesic, et al. [8]. Figure 1 shows seven basic gates of this multivalued switching system, where K-NAND, K-NOR, and K-M gates are defined by:

(1) K-NAND gate

$$z = K \text{ if } \min(x,y) = 0$$

$$= 0 \text{ otherwise}$$

(2) K-NOR gate

$$z = K \text{ if } \max(x,y) = 0$$

$$= 0 \text{ otherwise}$$

(3) K-M gate

$$z = K \text{ if } \frac{M}{x} = 0$$

$$= 0 \text{ otherwise}$$

For example, an and-OR realization of Eq. (4) is shown in Fig. 2

Algebra B

This algebra is defined the same as Algebra A except the m unary inverter operations and m unary cycling operations are replaced by the following operations:

(1) Let $c \in \{0, 1, 2, \dots, m-1\}$, the complement of c , denoted by \bar{c} , is defined as

$$\bar{c} = (m-1) - c$$

(2) Define a variable

$$\begin{matrix} a, b \\ x = m-1, \text{ if } a \leq x \leq b \\ = 0 \text{ otherwise} \end{matrix}$$

where $a, b \in \{0, 1, \dots, m-1\}$ and $a \leq b$. Note that

a, b
 x is a binary variable even though x is multivalued.

The complement of x , denoted by \bar{x} , is defined as

$$\begin{matrix} a, b & \bar{a}, \bar{b} \\ \bar{x} = 0, \text{ if } a \leq x \leq b \\ = m-1 \text{ otherwise} \end{matrix}$$

which is also two-valued. It should be noted that

$$\overline{a,b} \quad 0,a-1 \quad b+1,m-1$$

$$x = x + x$$

$$\text{where } x^{0,a-1} = 0 \text{ for } a = 0 \text{ and } x^{b+1,m-1} = 0 \text{ for } b = m-1.$$

It has been shown [9] that the sum, product, and complement operations of this algebra are a set of functionally complete operations, and any multivalued switching functions can be expressed by the sum-of-the-products canonical form:

$$f(x_1, \dots, x_n) = \sum_k c_k \cdot \overline{a_{k1}, b_{k1}}_{x_1} \cdot \overline{a_{k2}, b_{k2}}_{x_2} \cdot \dots \cdot \overline{a_{kn}, b_{kn}}_{x_n} \quad (5)$$

The complement of f is

$$\overline{f(x_1, \dots, x_n)} = \sum_k (\overline{c_k} + \overline{a_{k1}, b_{k1}}_{x_1} + \overline{a_{k2}, b_{k2}}_{x_2} + \dots + \overline{a_{kn}, b_{kn}}_{x_n}) \quad (6)$$

where $\overline{c_k} = (m-1) - c_k$ and $c_k \in \{0, 1, 2, \dots, m-1\}$.

Example 4: The canonical forms of the functions of Tables 1 and 2 are, respectively,

$$f(x, y) = 1 \cdot \begin{bmatrix} 0,0 & 0,0 & 2,2 & 0,0 \\ x \cdot y & + & x \cdot y \end{bmatrix} + 2 \cdot \begin{bmatrix} 0,0 & 2,2 & 2,2 & 2,2 \\ x \cdot y & + & x \cdot y \end{bmatrix} \quad (7)$$

$$f(x, y) = 1 \cdot \begin{bmatrix} 4,4 & 0,0 & 4,4 & 1,1 & 4,4 & 2,2 \\ x \cdot y & + & x \cdot y & + & x \cdot y \end{bmatrix} + \begin{bmatrix} 4,4 & 3,3 & 4,4 & 4,4 \\ x \cdot y & + & x \cdot y \end{bmatrix} + 2 \cdot \begin{bmatrix} 0,0 & 2,2 & 0,0 & 3,3 & 0,0 & 4,4 \\ x \cdot y & + & x \cdot y & + & x \cdot y \end{bmatrix} + \begin{bmatrix} 1,1 & 2,2 \\ x \cdot y \end{bmatrix} + 3 \cdot \begin{bmatrix} 0,0 & 0,0 & 1,1 & 3,3 \\ x \cdot y & + & x \cdot y \end{bmatrix} \quad (8)$$

A procedure for minimizing functions of the form of Eq. (5) was given by Su and Sarris [9]. For example, the function of Eq. (8) can be minimized by using a 5-valued map which is similar to the Karnaugh map as shown in Fig. 3. From this map it is found that the minimized function is

$$f(x, y) = 3 \cdot x \cdot y + 3 \cdot x \cdot y + 2 \cdot x \cdot y + 1 \cdot x$$

The set of basic operations can be realized by the circuit components shown in Fig. 4. The K-AND gate, K-OR gate, and Inverter of Fig. 4 can be implemented using conventional AND, OR, and NOT gates, respectively. The electronic implementation of the Digitizer remains to be investigated. As an example, the realization of the function of Eq. (7) using these basic circuit components is shown in Fig. 5.

Relationship between Algebra A and Algebra B

It is observed that Algebras A and B may be transformed from one to the other by utilizing the following relations between the inverter and cycling operations of Algebra A and the interval variable and the interval inverter operation of Algebra B: Let $1 \leq K \leq m-1$ and $1 \leq M \leq m-1$.

$$(1) \quad \overline{x}^M = K \cdot \overline{x}^{m-M, m-M} \quad (9)$$

$$(2) \quad \overline{x}^{m-M, m-M} \xrightarrow{M} \overline{x}^{m-1} \quad (10)$$

and

$$(3) \quad K \cdot \overline{x}^{m-M, m-M} = \overline{x}^M \cdot K \quad (11)$$

$$(4) \quad \left(\overline{x}_1^M + \overline{x}_2^M + \dots + \overline{x}_n^M \right)^K = K \cdot \overline{x}^{m-M_1, m-M_1} \cdot \overline{x}^{m-M_2, m-M_2} \cdot \dots \cdot \overline{x}^{m-M_n, m-M_n} \quad (12)$$

$$(5) \quad K \cdot \overline{x}^{a,b} = \overline{x}^{m-a} \cdot K + \overline{x}^{m-(a+1)} \cdot K + \dots + \overline{x}^{m-b} \cdot K \quad 0 \leq a < b < m-1 \quad (13)$$

For example, the functions of Eqs. (2) and (3) of Algebra A can be transformed to the functions of Eqs. (7) and (8) of Algebra B, respectively, by using Theorem 1 and Eq. (9).

III. THE PARTIAL DERIVATIVE

In this section we introduce the partial derivative of multivalued switching functions of Algebra A and Algebra B and their properties. First we define:

DEFINITION 2 Define the EXCLUSIVE-OR operation of two multivalued variables as

$$x \oplus y = x + y, \text{ if either } x \text{ or } y \text{ is zero.} \\ = 0, \text{ otherwise}$$

DEFINITION 3 Let $f(x_1, x_2, \dots, x_n)$ be a multivalued function of n variables x_1, x_2, \dots, x_n . The partial derivative is defined as:

$$\frac{\partial f(x_1, \dots, x_n, \overset{v_i}{x_i} k_i)}{\partial \overset{v_i}{x_i} k_i} = \left[f(x_1, \dots, x_n, \overset{v_i}{x_i} k_i) \right]^{m-1} \\ \oplus \left[f(x_1, \dots, x_n, \overset{v_i}{x_i} k_i) \right]^{m-1} \quad (14)$$

For Algebra B:

$$\frac{\partial f(x_1, \dots, x_n, k_i \cdot \overset{a_i, b_i}{x_i})}{\partial [k_i \cdot \overset{a_i, b_i}{x_i}]} \left[f(x_1, \dots, x_n, k_i \cdot \overset{a_i, b_i}{x_i}) \right]^{0,0} \\ \oplus \left[f(x_1, \dots, x_n, k_i \cdot \overset{a_i, b_i}{x_i}) \right]^{0,0} \quad (15)$$

The above definition is equivalent to the following definition.

DEFINITION 3'

$$\frac{f(x_1, \dots, x_n, \overset{v_i}{x_i} k_i)}{\overset{v_i}{x_i} k_i} = \left[f(x_1, \dots, x_n, m-1) \right]^{m-1} \\ \oplus \left[f(x_1, \dots, x_n, 0) \right]^{m-1} \quad (14')$$

$$\frac{f(x_1, \dots, x_n, k_i \cdot \overset{a_i, b_i}{x_i})}{k_i \cdot \overset{a_i, b_i}{x_i}} = \left[f(x_1, \dots, x_n, m-1) \right]^{0,0} \\ \oplus \left[f(x_1, \dots, x_n, 0) \right]^{0,0} \quad (15')$$

Example 5 The partial derivative of the function of Eq. (4) with respect to $\overset{1}{x}$ is

$$\frac{\partial f}{\partial [1 \cdot \overset{2,2}{x}]} = \left[x^1 \cdot y^1 + \overset{1}{x} \cdot y^1 + x^2 \cdot \overset{2}{y}^2 + \overset{2}{x}^2 \cdot \overset{2}{y}^2 \right]^2 \\ \oplus \left[x^1 \cdot y^1 + (\overset{1}{x})^1 y^1 + x^2 \cdot \overset{2}{y}^2 + \overset{2}{x}^2 \cdot \overset{2}{y}^2 \right]^2$$

$$\begin{array}{c|ccc} x & 0 & 1 & 2 \\ \hline y & 0 & 1 & 2 \\ \hline 0 & 0 & 2 & 0 \\ 1 & 2 & 2 & 2 \\ 2 & 0 & 2 & 0 \end{array} + \begin{array}{c|ccc} x & 0 & 1 & 2 \\ \hline y & 0 & 1 & 2 \\ \hline 0 & 0 & 0 & 2 \\ 1 & 2 & 2 & 2 \\ 2 & 0 & 2 & 0 \end{array} = \begin{array}{c|ccc} x & 0 & 1 & 2 \\ \hline y & 0 & 1 & 2 \\ \hline 0 & 0 & 2 & 2 \\ 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{array} \\ = \overset{2}{x}^2 \cdot y^2 + \overset{2}{x}^2 \cdot y^2 \quad (16)$$

Note that $\overset{1}{x}$ here denotes $\overset{1}{x}$.

The partial derivative of the function of Eq. (7) with respect to $1 \cdot \overset{2,2}{x}$ is

$$\frac{\partial f}{\partial [1 \cdot \overset{2,2}{x}]} = \left[\begin{array}{ccc} 0,0 & 0,0 & 2,2 & 0,0 \\ 1 \cdot x \cdot y + 1 \cdot x \cdot y + \end{array} \right]^{0,0} \\ \oplus \left[\begin{array}{ccc} 0,0 & 0,0 & 2,2 & 0,0 \\ 1 \cdot x \cdot y + 1 \cdot x \cdot y + \end{array} \right]^{0,0}$$

$\begin{array}{c} x \\ y \end{array}$	0	1	2
0	0	2	0
1	2	2	2
2	0	2	0

$\begin{array}{c} x \\ y \end{array}$	0	1	2
0	0	0	2
1	2	2	2
2	0	2	0

$\begin{array}{c} x \\ y \end{array}$	0	1	2
0	0	2	2
1	0	0	0
2	0	0	0

$$= 2 \cdot \overset{1,1}{x} \cdot \overset{0,0}{y} + 2 \cdot \overset{2,2}{x} \cdot \overset{0,0}{y} \quad (17)$$

Since Eqs. (4) and (7) both represent the same function whose truth table is described in Table I and by Eq. (9), the variable $\overset{1,1}{x}$ of Eq. (4) is equivalent to $\overset{2,2}{x}$ of Eq. (7), $\partial f / \partial \overset{1,1}{x}$ of Eq. (16) and $\partial f / \partial [\overset{2,2}{x}]$ of Eq. (17) produce the same result, as expected.

DEFINITION 4 As an extension of the single partial derivative of Eq. (14) the multiple partial derivative of a multivalued switching function $f(x_1, \dots, x_n)$ of Algebra A is:

$$\frac{\partial f(x_1, \dots, x_n, \overset{v}{x}_{i k_i}, \overset{v}{x}_{i k_i}, \overset{v}{x}_{k k_k})}{\partial (\overset{v}{x}_{i k_i}, \overset{v}{x}_{i k_i}, \overset{v}{x}_{k k_k})} = \left[f(x_1, \dots, x_n, \overset{v}{x}_{i k_i}, \overset{v}{x}_{i k_i}, \overset{v}{x}_{k k_k}) \right]^{m-1} \oplus \left[f(x_1, \dots, x_n, \overset{v}{x}_{i k_i}^{k_i}, \overset{v}{x}_{i k_i}^{k_i}, \overset{v}{x}_{k k_k}^{k_k}) \right]^{m-1} \quad (18)$$

Similar definition may be defined for Algebra B.

Several properties of the partial derivative are given below.

PROPERTY 1

$$\frac{\partial \bar{f}}{\partial \overset{v}{x}_{i k_i}} = \frac{\partial f}{\partial \overset{v}{x}_{i k_i}}$$

PROPERTY 2

$$\frac{\partial f}{\partial (\overset{v}{x}_{i k_i}^{k_i})} = \frac{\partial f}{\partial \overset{v}{x}_{i k_i}}$$

PROPERTY 3

$$\frac{\partial}{\partial \overset{v}{x}_{i k_i}} \left(\frac{\partial f}{\partial \overset{v}{x}_{i k_i}} \right) = \frac{\partial}{\partial \overset{v}{x}_{i k_i}} \left(\frac{\partial f}{\partial \overset{v}{x}_{i k_i}} \right)$$

PROPERTY 4

$$\frac{\partial (f \cdot g)}{\partial \overset{v}{x}_{i k_i}} = f \cdot \frac{\partial g}{\partial \overset{v}{x}_{i k_i}} \oplus g \cdot \frac{\partial f}{\partial \overset{v}{x}_{i k_i}}$$

$$\oplus \frac{\partial f}{\partial \overset{v}{x}_{i k_i}} \cdot \frac{\partial g}{\partial \overset{v}{x}_{i k_i}}$$

PROPERTY 5

$$\frac{\partial (f + g)}{\partial \overset{v}{x}_{i k_i}} = \bar{f} \cdot \frac{\partial g}{\partial \overset{v}{x}_{i k_i}} \oplus \bar{g} \cdot \frac{\partial f}{\partial \overset{v}{x}_{i k_i}}$$

$$\oplus \frac{\partial f}{\partial \overset{v}{x}_{i k_i}} \cdot \frac{\partial g}{\partial \overset{v}{x}_{i k_i}}$$

PROPERTY 6

$$\frac{\partial (f \oplus g)}{\partial \overset{v}{x}_{i k_i}} = \frac{\partial f}{\partial \overset{v}{x}_{i k_i}} \oplus \frac{\partial g}{\partial \overset{v}{x}_{i k_i}}$$

Similar relations may be derived for Algebra B.

IV. MACLAURIN AND TAYLOR SERIES EXPANSION

Based on the partial derivative of Definition 3, it is found that every multivalued switching function defined on Algebra A or Algebra B can be expressed in a MacLaurin series expansion and a Taylor series expansion. Before presenting the theorems, we need the following lemmas.

Lemma 1 Any multivalued function $f(x_1, \dots, x_n)$ can be expressed in the following form:

$$\begin{aligned}
f(x_1, \dots, x_n, \frac{y_i}{x_i} k_i) &= f(x_1, \dots, x_n, k_i \cdot \frac{y_i}{x_i}^{m-1}) \\
&= \frac{y_i}{x_i}^{m-1} \cdot f(x_1, \dots, x_n, k_i \cdot (m-1)) \\
&\quad + \left(\frac{y_i}{x_i}^{m-1} \right)^{m-1} \cdot f(x_1, \dots, x_n, k_i \cdot 0) \\
&= \frac{y_i}{x_i}^{m-1} \cdot \frac{y_i}{x_i}^{m-1} F^{m-1} + \left(\frac{y_i}{x_i}^{m-1} \right)^{m-1} \\
&\quad \cdot \frac{y_i}{x_i}^{m-1} F^0
\end{aligned} \quad (19)$$

Proof: Let $x = (x_1, \dots, x_n)$. Evaluating the function $F(X)$ of Eq. (19) for $\frac{y_i}{x_i}^{m-1} = m-1$, we have:

$$\begin{aligned}
f(x) \Big|_{\frac{y_i}{x_i}^{m-1} = m-1} &= (m-1) \cdot \frac{y_i}{x_i}^{m-1} F^{m-1} \\
+ 0 &= \frac{y_i}{x_i}^{m-1} F^{m-1}
\end{aligned}$$

Similarly, for $\frac{y_i}{x_i}^{m-1} = 0$,

$$\begin{aligned}
f(x) \Big|_{\frac{y_i}{x_i}^{m-1} = 0} &= 0 + (m-1) \cdot \frac{y_i}{x_i}^{m-1} F^0 \\
&= \frac{y_i}{x_i}^{m-1} F^0
\end{aligned}$$

Since $\frac{y_i}{x_i}^{m-1}$ can only be either 0 or $m-1$, Eq. (19) holds

for all values of $\frac{y_i}{x_i}^{m-1}$. This completes the proof.

Lemma 2 In view of Definition 2, Eq. (19) can be expressed as:

$$\begin{aligned}
f(x_1, \dots, x_n, \frac{y_i}{x_i} k_i) &= \frac{y_i}{x_i}^{m-1} \cdot f(x_1, \dots, x_n, k_i) \\
&\quad \oplus \frac{y_i}{x_i}^{m-1} \cdot f(x_1, \dots, x_n, 0) \\
&= \frac{y_i}{x_i} \cdot \frac{y_i}{x_i}^{m-1} F^{m-1} \oplus \left(\frac{y_i}{x_i} \right)^{m-1} \frac{y_i}{x_i} F^0
\end{aligned} \quad (20)$$

THEOREM 3 (MacLaurin Series)

The multivalued switching function can be expressed as:

For Algebra A

$$\begin{aligned}
f(x, y) &= \frac{\partial F}{\partial \frac{y}{y}^k} \Big|_{\frac{y}{y}^k = 0} \left(\frac{y}{y}^k \right)^0 \\
&\quad \oplus \frac{\partial F}{\partial \frac{y}{y}^k} \Big|_{\frac{y}{y}^k = 0} \left(\frac{y}{y}^k \right)^1 \oplus \dots \\
&\quad \oplus \frac{\partial F}{\partial \frac{y}{y}^{2p-1}} \Big|_{\frac{y}{y}^k = 0} \left(\frac{y}{y}^k \right)^{2p-1}
\end{aligned} \quad (21a)$$

For Algebra B

$$\begin{aligned}
f(x, y) &= \frac{\partial f}{\partial (k \frac{a}{y}^b)} \Big|_{k \frac{a}{y}^b = 0} \left(k \frac{a}{y}^b \right)^0 \\
&\quad \oplus \frac{\partial f}{\partial (k \frac{a}{y}^b)} \Big|_{k \frac{a}{y}^b = 0} \left(k \frac{a}{y}^b \right)^1 + \dots \\
&\quad \oplus \frac{\partial f}{\partial (k \frac{a}{y}^b)^{2p-1}} \Big|_{k \frac{a}{y}^b = 0} \left(k \frac{a}{y}^b \right)^{2p-1}
\end{aligned} \quad (21b)$$

where

$$\begin{aligned}
x &= (x_1, x_2, \dots, x_m) \\
y &= (y_1, y_2, \dots, y_p) \\
e &= (e_1, e_2, \dots, e_p) \quad 0 \in 2^{p-1}
\end{aligned}$$

$$y^e = y_1^{e_1} y_2^{e_2} \dots y_p^{e_p}$$

$$y^e = m-1, \text{ if } e=0$$

$$= y, \text{ if } e=1$$

$$\frac{y}{y}^k = \left(\frac{y_1}{y_1}^{k_1}, \dots, \frac{y_p}{y_p}^{k_p} \right)$$

$$\left(\frac{y}{y}^k \right)^e = \left(\frac{y_1}{y_1}^{k_1} \right)^{e_1} \left(\frac{y_2}{y_2}^{k_2} \right)^{e_2} \dots \left(\frac{y_p}{y_p}^{k_p} \right)^{e_p}$$

$$\frac{\partial f}{\partial (y^k)^e} = \frac{\partial}{\partial y_1^{k_1}} \left(\frac{\partial}{\partial y_2^{k_2}} \left(\dots \left(\frac{\partial f}{\partial y_p^{k_p}} \right) \dots \right) \right)$$

$$K = (k_0, k_1, \dots, k_{p-1})$$

$$\frac{a, b}{k y} = \left(\frac{a_1, b_1}{k_1 y_1}, \dots, \frac{a_p, b_p}{k_p y_p} \right)$$

$$\left(\frac{a, b}{k y} \right)^e = \left(\frac{a_1, b_1}{k_1 y_1} \right)^{e_1} \left(\frac{a_2, b_2}{k_2 y_2} \right)^{e_2} \dots \left(\frac{a_p, b_p}{k_p y_p} \right)^{e_p}$$

$$\text{and } \frac{\partial f}{\partial \left(\frac{a, b}{k y} \right)^e} = \frac{\partial}{\partial k_1^{a_1, b_1}} \left(\frac{\partial}{\partial k_2^{a_2, b_2}} \left(\dots \left(\frac{\partial f}{\partial k_p^{a_p, b_p}} \right) \dots \right) \right)$$

THEOREM 4 (Taylor Series)

A multivalued switching function can also be expressed as:

For Algebra A

$$f(x, y) = \frac{\partial f}{\partial y^k} \bigg|_{y^k=h} (y^k \oplus h)^0$$

$$\oplus \frac{\partial f}{\partial y^k} \bigg|_{y^k=h} (y^k \oplus h)^1$$

$$\oplus \dots \oplus \frac{\partial f}{\partial y^k} \bigg|_{y^k=h} (y^k \oplus h)^{2^p-1} \quad (22a)$$

For Algebra B

$$f(x, y) = \frac{\partial f}{\partial k^{a, b}} \bigg|_{k^{a, b}=h} (k^{a, b} \oplus h)^0$$

$$\oplus \frac{\partial f}{\partial k^{a, b}} \bigg|_{k^{a, b}=h} (k^{a, b} \oplus h)^1$$

$$\oplus \dots \oplus \frac{\partial f}{\partial k^{a, b}} \bigg|_{k^{a, b}=h} (k^{a, b} \oplus h)^{2^p-1} \quad (22b)$$

where $h = (h_1, h_2, \dots, h_p)$

$$(y \oplus h)^e = (y_1 \oplus h_1)^{e_1} (y_2 \oplus h_2)^{e_2} \dots (y_p \oplus h_p)^{e_p}, 0 \leq e \leq 2^p-1$$

Theorems 3 and 4 are best illustrated by examples.

Example 6 Find the MacLaurin series expansion of the following 3-valued switching function:

$$f(x_1, x_2, y_1, y_2) = \frac{2}{x_1^2} \cdot \frac{2}{y_2^2} + x_2^2 \cdot \frac{1}{y_2^2} \quad (23)$$

To obtain the series expansion, we need to compute the following three partial derivatives:

$$(1) \frac{\partial f}{\partial x_1^2}, (2) \frac{\partial f}{\partial y_2^2}, \text{ and } (3) \frac{\partial}{\partial y_1^2} \left(\frac{\partial f}{\partial y_2^2} \right)$$

They are computed as follows:

$$(1) \frac{\partial f}{\partial x_1^2} = \left(\frac{2}{x_1^2} \cdot \frac{1}{y_1^2} + x_2^2 \cdot \frac{1}{y_2^2} \right)^2$$

$$\oplus \left(\frac{1}{x_1^2} \cdot \left(\frac{2}{y_1^2} \right)^2 + x_2^2 \cdot \frac{1}{y_2^2} \right)^2$$

$$= \left(\frac{2}{x_1^2} + x_2^2 \cdot \frac{1}{y_2^2} \right)^2 \oplus \left(x_2^2 \cdot \frac{1}{y_2^2} \right)^2$$

x_1, x_2 y_2	00	01	02	10	11	12	20	21	22
0	2	2	2	0	0	0	2	2	2
1	2	2	2	0	0	0	2	2	2
2	0	2	2	0	0	0	0	2	2

$x_1 x_2$ y_2	00	01	02	10	11	12	20	21	22
0	2	2	2	2	2	2	2	2	2
$\oplus 1$	2	2	2	2	2	2	2	2	2
2	0	2	2	0	2	2	0	2	2

$x_1 x_2$ y_1	00	01	02	10	11	12	20	21	22
0	2	2	2	2	2	2	2	2	2
$\oplus 1$	2	2	2	0	0	0	2	2	2
2	2	2	2	2	2	2	2	2	2

$x_1 x_2$ y_2	00	01	02	10	11	12	20	21	22
0	0	0	0	2	2	2	0	0	0
$= 1$	0	0	0	2	2	2	0	0	0
2	0	0	0	0	2	2	0	0	0

$x_1 x_2$ y_1	00	01	02	10	11	12	20	21	22
0	2	0	0	2	0	0	2	0	0
$= 1$	2	0	0	0	0	0	2	0	0
2	2	0	0	2	0	0	2	0	0

$$\frac{\partial f}{\partial y_1^2} = \frac{2}{x_1^2} y_2^2 + \frac{2}{x_1^2} \frac{2}{y_2^2} + \frac{2}{x_1^2} \frac{2}{x_2^2} + \frac{2}{x_1^2} \frac{1}{x_2^2}$$

$$= \frac{2}{x_1^2} \left(\frac{1}{y_2^2} \right)^2 + \frac{2}{x_1^2} \frac{2}{x_2^2} + \frac{2}{x_1^2} \frac{1}{x_2^2}$$

$$\frac{\partial f}{\partial y_2^2} \left| \begin{array}{l} \frac{2}{y_1^2} = 0 \\ \frac{1}{y_2^2} = 0 \end{array} \right. = \frac{2}{x_1^2} + \frac{2}{x_1^2} \frac{2}{x_2^2} + \frac{2}{x_1^2} \frac{1}{x_2^2} = \frac{2}{x_1^2}$$

$$(2) \frac{\partial f}{\partial \frac{1}{y_2^2}} = \left(\frac{2}{x_1^2} \frac{2}{y_1^2} + x_2^2 \frac{1}{y_2^2} \right)^2$$

$$\oplus \left(\frac{2}{x_1^2} \frac{2}{y_1^2} + x_2^2 \left(\frac{1}{y_2^2} \right)^2 \right)^2$$

$$= \left(\frac{2}{x_1^2} \frac{2}{y_1^2} + x_2^2 \right)^2 \oplus \left(\frac{2}{x_1^2} \frac{2}{y_1^2} \right)^2$$

$x_1 x_2$ y_1	00	01	02	10	11	12	20	21	22
0	0	2	2	0	2	2	0	2	2
$= 1$	0	2	2	0	0	0	0	2	2
2	0	2	2	0	2	2	0	2	2

$$\frac{\partial f}{\partial \frac{1}{y_2^2}} = x_1^2 x_2^2 + \frac{1}{x_1^2} x_2^2 + x_2^2 y_1^2 + x_2^2 \frac{1}{y_1^2}$$

$$= x_1^2 x_2^2 + \frac{1}{x_1^2} x_2^2 + x_2^2 \left(\frac{2}{y_1^2} \right)^2$$

$$\frac{\partial f}{\partial y_2^2} \left| \begin{array}{l} \frac{2}{y_1^2} = 0 \\ \frac{1}{y_2^2} = 0 \end{array} \right. = x_1^2 x_2^2 + \frac{1}{x_1^2} x_2^2 + x_2^2 = x_2^2$$

$$(3) \frac{\partial}{\partial \frac{1}{y_2^2}} \left(\frac{\partial f}{\partial \frac{2}{y_1^2}} \right) = \left(\frac{2}{x_1^2} \left(\frac{1}{y_2^2} \right)^2 \right.$$

$$\left. + \frac{2}{x_1^2} \frac{2}{x_2^2} + \frac{2}{x_1^2} \frac{1}{x_2^2} \right)^2$$

$$\oplus \left(\frac{2}{x_1^2} \frac{1}{y_2^2} + \frac{2}{x_1^2} \frac{2}{x_1^2} + \frac{2}{x_1^2} \frac{1}{x_2^2} \right)^2$$

$$= \left(\frac{2}{x_1^2} \right)^2 \oplus \left(\frac{2}{x_1^2} \frac{2}{x_2^2} + \frac{2}{x_1^2} \frac{1}{x_2^2} \right)^2$$

x_1 x_2	0	1	2
0	2	0	2
$= 1$	2	0	2
1	2	0	2

x_1 x_2	0	1	2
0	2	2	2
$\oplus 1$	2	0	2
2	2	0	2

x_1 x_2	0	1	2
0	0	2	0
$= 1$	0	0	0
2	0	0	0

$$\frac{\partial}{\partial \frac{1}{y_2^2}} \left(\frac{\partial f}{\partial \frac{2}{y_1^2}} \right) = \frac{2}{x_1^2} x_2^2$$

Therefore,

$$f(x_1, x_2, y_1, y_2) = f(x_1, x_2, 0, 0) \oplus \frac{\partial f}{\partial \frac{2}{y_1^2}} \bigg|_{\substack{\frac{2}{y_1^2} = 0 \\ \frac{1}{y_2^2} = 0}} \left(\frac{2}{y_1^2} \right)$$

$$\oplus \frac{\partial f}{\partial \frac{1}{y_2^2}} \bigg|_{\substack{\frac{2}{y_1^2} = 0 \\ \frac{1}{y_2^2} = 0}} \left(\frac{1}{y_2^2} \right)$$

$$\oplus \frac{\partial}{\partial \frac{1}{y_2^2}} \left(\frac{\partial f}{\partial \frac{2}{y_1^2}} \right) \bigg|_{\substack{\frac{2}{y_1^2} = 0 \\ \frac{1}{y_2^2} = 0}} \left(\frac{2}{y_1^2} \right) \left(\frac{1}{y_2^2} \right)$$

$$= 0 \oplus \frac{2}{y_1^2} \frac{2}{y_1^2} \oplus \frac{1}{y_2^2} x_2^2 \oplus \frac{2}{y_1^2} \frac{1}{y_2^2} \frac{2}{x_1^2} x_2^2$$

V. APPLICATION TO FAULT DETECTION OF MULTIVALUED SWITCHING CIRCUITS

As an example of demonstrating the potential usefulness of the partial derivative of multivalued switching function of Definition 3, the derivation of fault-detection tests of multivalued combinational circuits using it is presented. First, for illustration convenience, we define:

DEFINITION 7

The two multivalued switching systems defined by Algebra A and Algebra B will be referred to as System A and System B, respectively.

Unlike a binary switching circuit, a multivalued switching circuit, even it is irredundant, may have undetectable faults. This is described in the following theorem.

Theorem 5 The faults s-a-k, for $K < k < m-1$ of a K-NAND (K-AND) gate of System A (System B) are undetectable. Dually, the faults s-a-k, for $0 < k < K$ of a K-NOR (k-OR) gate of System A (System B) are undetectable.

The proof of this theorem is obvious and may thus be omitted.

The following two theorems show that the tests for detecting any detectable faults in a multivalued combinational circuit can be obtained by using the partial derivative of the output function of the circuit.

Theorem 6

Let $f(x_1, \dots, x_n)$ be the output function of a combinational circuit. (a_1, \dots, a_n)

a_i $0, 1, 2, \dots, m-1$ is a test of s-a-0 or s-a-k, $1 < k < K$

fault line j if and only if

For System A

$$\frac{\partial f}{\partial \left(\frac{M}{x_i} \right)} \bigg|_{a_1, \dots, a_n} = (m-1) \quad (24a)$$

For System B

$$\frac{\partial f}{\partial \left(K \cdot x_i \right)} \bigg|_{a_1, \dots, a_n} = (m-1) \quad (24b)$$

Theorem 7

(a) (a_1, \dots, a_n) is a test if s-a-0 fault on line j if and only if

For System A

$$\frac{M}{x_i} \cdot \frac{\partial f}{\partial \left(\frac{M}{x_i} \right)} \bigg|_{a_1, \dots, a_n} = K \quad (25a)$$

For System B

$$a_i \cdot \frac{\partial f}{\partial \left(\frac{a_i}{x_i} \right)} \bigg|_{a_1, \dots, a_n} = K \quad (25b)$$

(b) (a_1, \dots, a_n) is a test of s-a-k fault on line j , $1 < k < K$, if and only if

For System A

$$\left(\frac{M}{x_i} \right)^K \frac{\partial f}{\partial \left(\frac{M}{x_i} \right)} \bigg|_{a_1, \dots, a_n} = K \quad (26a)$$

For System B

$$K \frac{\partial f}{\partial x_i} \bigg|_{a_1, \dots, a_n} = K \quad (26b)$$

Multiplying $\frac{M}{x_i} K \left[K \cdot x_i^{a,b} \right]$ to Eq. (24a) Eq. (24b) would

make all the terms of Eq. (24a) Eq. (24b) having x_i with different values of M vanish, and the remaining terms of Eq. (25a) Eq. (25b) constitute T_0 . Similarly,

multiplying $\left(\frac{M}{x_i} K \right) \left[K \cdot x_i^{a,a} \right]$ to Eq. (24a) Eq. (24b)

would make all the terms of Eq. (24a) Eq. (24b) having x_i with the same value of M [a] vanish, and the remaining terms of Eq. (26a) Eq. (26b) constitute T_k .

The following example illustrates these theorems.

Example 9 Consider the 3-valued combinational circuit of Fig. 2, which for convenience is repeated in Fig. 7. Suppose it is desired to derive a set of tests for detecting s-a-0, s-a-1, and s-a-2 faults on line indicated in the figure. Note that this line is connected to the output of a K-M gate with $K = 1$.

The output function of this circuit was given in Eq. (4) and its partial derivative with respect to \vec{x}^1 was described in Eq. (14) which is

$$\frac{\partial f}{\partial \vec{x}^1} = \vec{x}^2 \cdot y^2 + \frac{2}{x} 2 \cdot y^2 \quad (14)$$

According to Theorem 4, the set T of tests for detecting s-a-0 and s-a-1 faults on line j are the values of x and y satisfying the following equation:

$$\frac{\partial f}{\partial \vec{x}^1} = \vec{x}^2 \cdot y^2 + \frac{2}{x} 2 \cdot y^2 = 2 \quad (27)$$

It is found that when $x=1, y=0$ and $x=2, y=0$ the above equation is satisfied. Thus,

$$T = \{10, 20\}$$

Moreover, from Theorem 5, the sets T_0 and T_1 of tests for detecting s-a-0 and s-a-1 faults on line j , respectively, can be obtained from the following equations:

$$\begin{aligned} \vec{x}^1 \cdot \frac{\partial f}{\partial \vec{x}^1} &= \vec{x}^1 \cdot \vec{x}^2 \cdot y^2 + \vec{x}^1 \cdot \frac{2}{x} 2 \cdot y^2 = \vec{x}^1 \cdot y^2 \\ &= 1 \end{aligned} \quad (28a)$$

$$\left(\frac{\vec{x}^1}{x} \right) \frac{\partial f}{\partial \vec{x}^1} = (\vec{x}^1)^1 \cdot \vec{x}^2 \cdot y^2 + (\vec{x}^1)^1 \cdot \frac{2}{x} 2 \cdot y^2 = (\vec{x}^1)^1 \quad (28b)$$

$$\frac{2}{x} 2 \cdot y^2 = 1$$

which indicate

$$T_0 = \{20\} \quad (29a)$$

and

$$T_1 = \{10\} \quad (29b)$$

Several remarks should be made here.

- (1) Notice that $T = T_0 \cup T_1$, as expected.
- (2) The tests 20 and 10 do sensitize the path from \vec{x}^1 to the circuit output z as shown in Fig. 6, as they should.
- (3) Faults s-a-1 and s-a-2 on line j are detectable, but indistinguishable.
- (4) These three sets of tests, T , T_0 , and T_1 , may be obtained from Theorem 4 and 5 (for System B) and Eq. (15) for detecting s-a-0 and s-a-1 faults at the corresponding location in the combinational circuit of Fig. 5.

VI. CONCLUSION

Boolean differential calculus has been generalized to multivalued algebras. Two potentially implementable multivalued algebras and the partial derivatives of multivalued switching functions defined on them have been studied. Simple relations between the two algebras are presented. By using them, one can convert one system from the other. Furthermore, it has been shown that every function of these algebras can be expanded into a MacLaurin series and a Taylor series. Examples illustrating the procedures for obtaining these series expansions are given. As an example of the application of the theory, the test derivation for detecting a fault in a multivalued combinational circuit is discussed. The results of further investigation of the applications of digital calculus will be given in a subsequent paper.

VII. REFERENCES

In the following references, STAMLD stands for the Symposia on the Theory and Applications of Multiple-valued Logic Design and ISML for the International Symposia on Multiple-valued Logic.

1. Reed, I.S.: A class of multiple-error-correcting codes and the decoding scheme. IRE Trans. IT-4 (1954), pp. 38-49.
2. Akers, S.B., Jr.: On a theory of Boolean functions J. Soc. Indust. Appl. Math. Vol. 7, No. 4, December, 1959, pp. 487-498.
3. Sellers, F.F.; Hsiao, M. Y.; Bearnson, L.W.: Analyzing errors with the Boolean difference. IEEE Trans. Comput. Vol. C-17, July 1968, pp. 676-683.
4. Thayse, A.: Boolean differential calculus Philips Res. Repts., 26, 1971, pp. 229-246.
5. Lee, S.C.; Keren-Zvi, Y.: A generalized Boolean Algebra and its applications to logic design. 1975 ISML, pp. 88-98.
6. Lee, S.C.: Vector Boolean algebra and calculus, IEEE Trans. on Computers. Vol. C-25, September 1975, pp. 865-874.
7. Lee, S.C.: Modern Switching Theory and Digital Design, Prentice-Hall, to appear in January 1978.
8. Vranesic, Z.; Lee, E.; and Smith, K.C.: A many valued algebra for switching systems. IEEE Trans. on Computers, Oct. 1970, pp. 964-971.
9. Su, S.Y.H.; Sarris, A.A.: The Relationship Between Multi-valued Algebra and Boolean Algebra under Different Definitions of Complement IEEE Trans. Comput., vol C-21, May 1972, pp. 479-485.
10. Post, E.: Introduction to a general theory of elementary propositions. Amer. J. Math. 93 (1921), 163-185. Reprinted: From Frege to Godel (J. van Hiejenroot, ed.), Cambridge, Mass. 1967.
11. Brzozowski, J.A.; Yoeli, M.: Digital Networks, Prentice Hall, 1975.
12. Lee, S.C.: Digital Circuits and Logic Design, Prentice-hall, 1976.
13. Marinos, P.N.: Derivation of minimal complete sets of test-input sequences using Boolean differences. IEEE Trans. Comput., Vol. C-20, Jan. 1971.
14. Marinos, P.; Page, E.; Thomason, M.: Fult detection in sequential networks using time dependent Boolean differences. 1971 Proc. IEEE Int. Comput. Soc. Conf., pp. 79-80.
15. Rescher, N.: Many-valued logic. McGraw-Hill, N.Y., 1969.
16. Rine, D.C., (ed.): Computer Science and Multi-valued Logic, North-Holland, 1977.
17. Sheppard, D.A.: An application of many-valued logics to fault detection. 1973 ISML, pp. 192-204.
18. Thayse, A.: A variational diagnosis method for stuck-faults in combinational networks Philips Res. Repts, 27, 1972, pp. 82-98.
19. Thayse, A.: Disjunctive and conjunctive operators for Boolean functions. Philips Res. Repts., 28, 1973, pp. 1-16.
20. Thayse, A.; Davio, M.: Boolean differential calculus and its application to switching theory, IEEE Trans. on Computers, Vol. C-22, 1973, pp. 499-420.
21. Thayse, A.: Differential calculus for functions from $(GF(p))^n$ into $GF(p)$, Philips Res. Repts. 29 (1974), pp. 560-586.
22. Vranesic, Z.; Smith, K.C.: Engineering aspects of multi-valued logic systems. Computer, Sept. 1974, pp. 34-41.
23. Vranesic, Z.; Waliuzzaman, K.M.: Functional transformation in simplification of multivalued switching functions, IEEE Trans. Comput., Jan. 1972, pp. 102-105.
24. Wojcik, A.C.: The Minimization of higher-order Boolean functions. 1972 STAMLD, pp. 181-192.
25. Wojcik, A.C.; Metze, C.: Some relationships between Post and Boolean algebras. 1971 STAMLD, pp. 173-182.
26. Wolf, R.G.: A critical survey of many-valued logics 1966-1974. 1975 ISML, pp. 468-474.
27. Yau, S.S.; Tang, Y.S.: An efficient algorithm for generating complete test sets for combinational logic circuits. IEEE Trans. Comput., Vol. C-20, Nov. 71, pp. 1245-1251.

TABLE 1

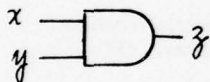
TRUTH TABLE OF THE FUNCTION OF EXAMPLE 1

x	0	0	0	1	1	1	2	2	2
y	0	1	2	0	1	2	0	1	2
f(x,y)	1	0	2	0	0	0	1	0	2

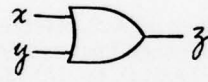
TABLE 2

TRUTH TABLE OF THE FUNCTION OF EXAMPLE 2

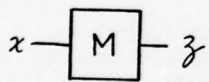
x	0	0	0	0	1	1	4	4	4	4	4	otherwise
y	0	2	3	4	2	3	0	1	2	3	4	
f(x,y)	3	2	2	2	2	3	1	1	1	1	1	



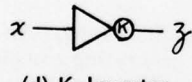
(a) AND gate



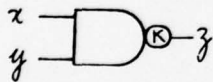
(b) OR-gate



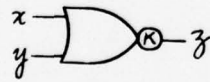
(c) M-cycling gate



(d) K-Inverter



(e) K-NAND gate



(f) K-NOR gate



(g) K-M gate

Fig. 1 Seven basic gates of the multivalued switching system derived from Algebra A.

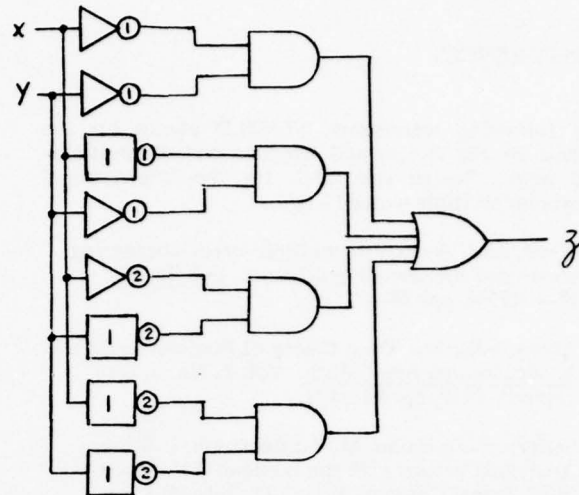
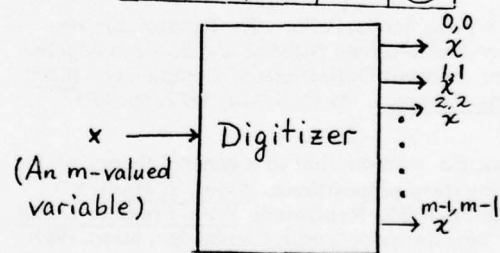
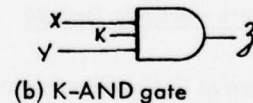


Fig. 2 AND-OR realization of Eq. (4).

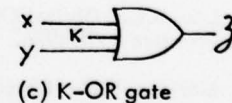
y \ x	0	1	2	3	4
0	(3)				(1)
1					(1)
2	(2)	(2)			(1)
3	(2)	(2)			(1)
4	(3)				(1)



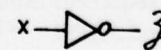
(a) Digitizer



(b) K-AND gate



(c) K-OR gate



(d) Inverter

Fig. 4. Four basic circuit components of the multivalued switching system derived from Algebra B.

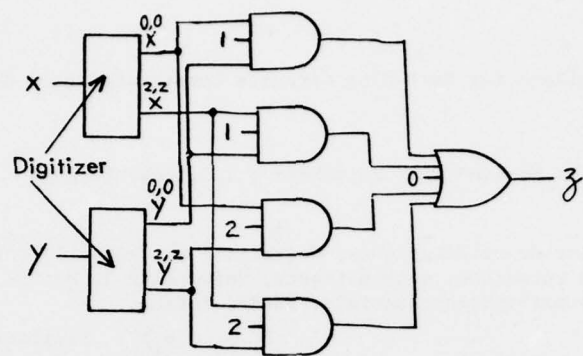
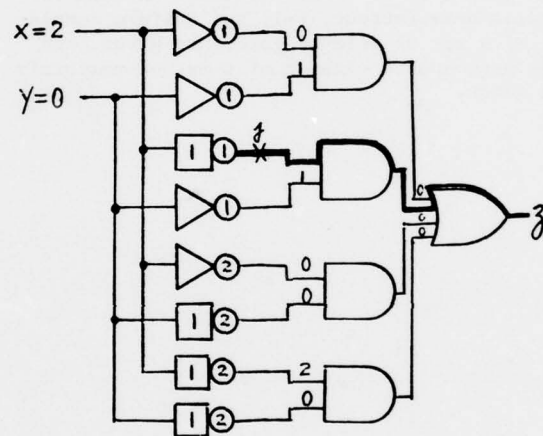
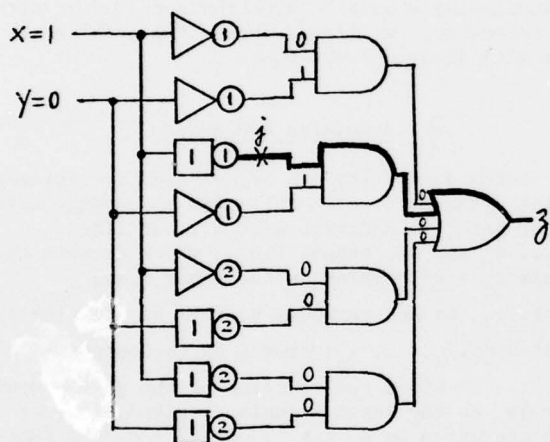


Fig. 5. AND-OR realization of the function of Eq. (7)



Testing for s-a-1 and s-a-2 faults		
Test $x \ y$	Value on line j	Value of z
1 0	0 (Normal)	0 (Normal)
	1 (Faulty)	1 (Faulty)

Testing for s-a-0 fault		
Test $x \ y$	Value on line j	Value of z
2 0	1 (Normal)	1 (Normal)
	0 (Faulty)	0 (Faulty)

Fig. 6 - An example of fault detection in multivalued combinational circuits

L. Martin* ; C. Reischer* ; I.G. Rosenberg**

* Département de mathématiques, Université du Québec à Trois-Rivières

** Centre de recherches mathématiques, Université de Montréal and
Centre de mathématique sociale, EHESS, Paris

The capability of the synthesis of finite deterministic automata build up from delayed gates over a finite alphabet \underline{d} is studied by reduce that problem to the problem of building circuits realizing functions of several variables ranging over \underline{d} and with values in \underline{d} . Some characterizations for the completeness (strong, full and uniform completeness) of a set of delayed gates are given, and also for precomplete classes of a set of uniformly delayed gates.

b_1, \dots, b_i)). Similarly the output function is represented by e functions of $s+i$ variables. This brings us to the problem of building circuits realizing functions of several variables ranging over \underline{d} and with values in \underline{d} based on the gates which are at our disposal (e.g. available commercially). Note that we have to realize several circuits whose functioning should be synchronized within certain tolerance. We discuss here this problem for gates with delayed reactions.

§ 2. \underline{d} -modules and \underline{d} -nets

Let \mathbb{N} denote the set of non-negative integers and let $d \in \mathbb{N}$ be fixed. For $n \in \mathbb{N}$ a \underline{d} -module (element or gate) is a device with n input lines $\varepsilon_1, \dots, \varepsilon_n$ and one output line ω which accepts the signals $\chi_i^n \in \underline{d}$ incoming on the input lines ε_i ($i=1, \dots, n$) and responds to them by emitting the signal $y = f(\chi_1, \dots, \chi_n)$ from \underline{d} on the output line ω .

Since the reaction of the gate depends entirely on the input signals received, f is an n -ary operation on \underline{d} i.e. a map $\underline{d}^n \rightarrow \underline{d}$ or a function with n arguments and values ranging over \underline{d} . (The case of a partial function is more complex due to prohibited configurations and is not discussed here). For a later reference we denote the set of all n -ary operations on \underline{d} by $\Theta^{(n)}$ (hence the size

$|\Theta^{(n)}|$ of $\Theta^{(n)}$ is $d^{(d^n)}$ and set $\Theta = \Theta^{(1)} \cup \Theta^{(2)} \cup \dots$. In universal algebra it is customary to use also the nullary or zero operations corresponding to the inputless \underline{d} -modules (i.e. the emitters of constant signals or constant sources which in practice are mostly available at no or little cost); however for our purposes we replace them by constant one variable operations. Thus each \underline{d} -module is an automaton by itself and the totality of the available types of \underline{d} -modules represents the building blocks at our disposal. We emphasize that for us the \underline{d} -modules are simply "black boxes" in the sense that they are further indecomposable, we do not care what is happening inside them and all what matters and is known to us is their behavior which, for example, can be described by a table with d^n rows corresponding to the n -tuples $(x_1, \dots, x_n) \in \underline{d}^n$ of arguments and the values $f(x_1, \dots, x_n)$. We pause briefly to stress that we

§ 1. Introduction

The topic of this paper is the capability of the synthesis of automata built up from delayed gates. The automata studied are finite, deterministic but not necessarily completely defined and are given e.g. by their transition digraphs whose vertices are the states and whose arrows are labelled by the pairs consisting of the input causing the transition (from a state where the arrow starts to the state it ends) and by the output produced. We put ourselves in the situation when the available gates from which the concrete realization of the automaton is to be constructed all work in the same finite alphabet which for notational convenience is denoted $\underline{d} = \{0, 1, \dots, d-1\}$. To get a realization first we code (i.e. address) the states inputs and outputs by elements from \underline{d}^s , \underline{d}^i , and \underline{d}^e , respectively (where s, i, e are positive integers and \underline{d}^n denotes the set of all n -tuples of elements from \underline{d}). Basing ourselves on the chosen coding the transition function can be expressed by s functions (possibly only partially defined) of $s+i$ variables ranging over \underline{d} and whose values are all in \underline{d} (i.e. to the state \bar{s} coded by (a_1, \dots, a_s) and input I coded by (b_1, \dots, b_i) we assign the state $(f_1(a_1, \dots, a_s, b_1, \dots, b_i), \dots, f_s(a_1, \dots, a_s,$

are interested in feasibility rather than optimality here. Thus typically we ask what circuits can be built from given d-modules and ignore completely the problem of doing it in an economical way. This limitation has a good reason because the optimality depends on the costs largely determined by the present technology and labor costs and should therefore be closely tailored to a very specific situation. The technological changes incurred during the past few decades indeed indicate that one should think twice before undertaking some basic theoretical optimization work. Our disregard for costs also explains the above carelessness in the addressing of states, inputs and outputs d-modules (the later based on the theoretical possibility of replacing one k output d-module by k one output d-modules).

Practically in every physical realization of d-modules there is a certain time delay in the response to a change in input signals. Sometimes this delay is so short that it may be neglected, say, by setting the clock (timing the signals coming from the outside) so that the basic time interval Δt (during which the outside signals are invariant i.e. they may change only at, or around, the times $0, \Delta t, 2\Delta t, \dots$) far outweighs the delay. We are not considering here the d-modules that during their switching become momentarily "deaf" to any subsequent change of inputs (i.e. stay stable for a certain period following the switch). However, the very presence of and the problems caused by the races, hazards etc. in the real circuits indicate that the total neglect of delays is an oversimplification. Moreover, to increase the flow of information on the channels, Δt should be as small as possible. With these rather loose comments in mind we look into the problems of constructing circuits from delayed d-modules. A delayed d-module is a d-module whose output at the time t is

$$x(t) = f(x_1(t-\delta_1), \dots, x_n(t-\delta_n))$$

where $f \in \mathcal{O}^{(n)}$ and $\delta_1, \dots, \delta_n$ are time invariant delays (the delayed modules are called delayed input devices in [7]). For simplicity all δ_i 's are assumed to be non negative integers (if they are rational and finite in number this clearly is no real restriction). Thus we may identify a delayed d-module F with the $(n+1)$ -tuple $(f, \delta_1, \dots, \delta_n)$. Then the set $D^{(n)}$ of all delayed n -ary d-modules is $\mathcal{O}^{(n)} \times \mathbb{N}^n$. We set $D = D^{(1)} \cup D^{(2)} \cup \dots$

In circuits as well as in neuron sets etc. the d-modules are combined together by attaching the output of one module to the input of another module. An aggregate of d-modules interconnected in this way is called a d-modular net. Thus a d-modular net is an arrangement of d-modules in which the output (input) of each module is connected to at most one input (output) of another module (the restriction at most one being obtained by possible duplication). The outputs (inputs) not connected to any input (output) are called the external out-

puts (inputs). The external inputs are labelled by input variables (possibly several by the same input variable). These interconnections can be captured by a partition of the set consisting of all inputs and outputs of the d-modules ([9]). An intuitive and pictorial way is to represent it by a rooted oriented forest. In it each of the trees whose disjoint union makes up the forest is oriented towards its root, its vertices consist of d-modules and external inputs and there is an arc joining a vertex to another if the output of the d-module or the external variable corresponding to the first is connected to the input of the second. Thus the roots correspond to having an external output and the leaves are the outside inputs (labelled by the input variables).

At this stage the reader may be wondering where the feedbacks got lost. Their disappearance is due to the mental ledgerdemon of splitting the multiple output d-modules into the single output ones (and trivially discarding those trees having no external outputs). Obviously we may consider each tree separately. The study of such trees is nothing else than the analysis of the composition of delayed operations which we describe now in detail. The precise definition being somewhat cumbersome at any rate, we proceed directly to the algebraic definition based on the ideas of A.I. Mal'cev [8]. The composition in D is conceived as the result of applying certain operations on D (i.e. in some sense we operate on a higher level or have a "metatheory"). More precisely, we introduce the following unary operations (selfmaps) $\zeta, \tau, \Delta, \pi_i$ ($i \in \mathbb{N}$) and one binary operation $*$ on D .

For $F \in D^{(1)}$ set $\zeta(F) = \tau(F) = F$ while for $F = (f, \delta_1, \dots, \delta_n) \in D^{(n)}$ ($n > 1$) set $\zeta(F) = (\zeta f, \delta_2, \dots, \delta_n, \delta_1)$, $\tau(F) = (\tau f, \delta_2, \delta_1, \delta_3, \dots, \delta_n)$ where $\zeta(f)(x_1, \dots, x_n) = f(x_2, \dots, x_n, x_1)$ and $\tau(f)(x_1, \dots, x_n) = f(x_2, x_1, x_3, \dots, x_n)$ for all $x_1, \dots, x_n \in \underline{d}$. Thus ζ is devised to cyclically permute the inputs and τ to interchange the two first inputs. From the elementary theory of permutations it follows that the repeated application of ζ and τ can produce any input permutation. The identification of two inputs is feasible only if the corresponding delays are identical. Hence we define Δ by setting (i) $\Delta(F) = F$ if in $F = (f, \delta_1, \dots, \delta_n)$ either $n = 1$ or $\delta_1 \neq \delta_2$, and (ii)

$$\Delta(f, \delta_1, \delta_1, \delta_2, \dots, \delta_{n-1}) = (\Delta f, \delta_1, \delta_2, \dots, \delta_{n-1})$$

otherwise (where $(\Delta f)(x_1, \dots, x_{n-1}) = f(x_1, x_1, x_2, \dots, x_{n-1})$ for all $x_1, \dots, x_{n-1} \in \underline{d}$). The repeated application of ζ, τ and Δ yields any permitted change of variables (reindexing with allowable identifications). The operations π_i ($i \in \mathbb{N}$) are devised to alter the delays of dummy variables. We say that the i -th variable of $F = (f, \delta_1, \dots, \delta_n)$

is dummy if $f(a_1, \dots, a_n) = f(b_1, \dots, b_n)$ whenever $a_j = b_j$ for all $j \neq i$. Clearly the delay in a dummy variable is irrelevant. We set $\kappa_i(F) = (f, i, \delta_2, \dots, \delta_n)$ if the first variable is dummy and $\kappa_i(F) = F$ otherwise. The presence of κ_i 's permits to change arbitrary the delay of a dummy variable of an operation obtained via composition. This flexibility was not adopted in [1, 3, 4] leading in the extreme to the formal distinction between constant functions with different delays (anyway the constant functions are usually of minor importance in applications because they are available at little or no cost). Another and equivalent approach is to right away identify F and G differing in the delay of a dummy variable only (or going ever further not allow dummy variables altogether). To avoid certain formal difficulties this approach is not adopted here.

For the composition itself we have the binary operation $*$ defined as follows. Let $F = (f, \delta_1, \dots, \delta_m) \in D$ and $G = (g, \lambda_1, \dots, \lambda_n) \in D$. Set

$$F * G = (f * g, \delta_1 + \lambda_1, \dots, \delta_1 + \lambda_n, \delta_2, \dots, \delta_m)$$

where

$$(f * g)(x_1, \dots, x_{m+n-1}) = f(g(x_1, \dots, x_m), x_{m+1}, \dots, x_{m+n-1})$$

for all $x_1, \dots, x_{m+n-1} \in d$. Thus $F * G$ replaces the first variable in f by g while compounding the corresponding delays. The repeated application of $*$, ζ , τ , κ and Δ can produce all compositions.

For $M \subseteq D$ let $[M]$ denote the least subset of D containing M and closed with respect to the formation of ζ , τ , Δ , κ_i ($i \in \mathbb{N}$) and the products $*$ (i.e. the subalgebra of $\langle D; \zeta, \tau, \Delta, \kappa_i (i \in \mathbb{N}), * \rangle$ generated by M). It is not difficult to convince oneself that $[M]$ is nothing else than the set of delayed operations which can be constructed from the delayed modules from M via tree shaped d-modular nets. In other words, if the market can supply us with d-modules from M (in potentially unlimited quantities), then $[M]$ consists exactly of the delayed operations that can be constructed (regardless of costs) in all possible ways from the d-modules from M .

The simplest delayed d-modules are those having the same delay of all variables. We call them uniformly delayed d-modules (the papers [1, 3, 4] dealt exclusively with the uniformly delayed 2-modules and they are called delayed output device in [7]). For $i \in \mathbb{N}$ set

$$U_i = \{(f_1, \delta_1, \dots, \delta_n) \in M : \delta_1 = \dots = \delta_n = i\}.$$

Clearly $U = U_0 \cup U_1 \cup \dots$ is the set of all uniformly delayed d-modules. We say that $F \in \Theta$ is complete (other names : functionally complete or $\langle d; F \rangle$ primal) if each $f \in \Theta$ is obtainable from

F via repeated (ordinary) composition (i.e. Θ is the least subset of Θ containing F and closed with respect to the operations ζ , τ , Δ and $*$ in Θ defined earlier (completeness in Θ has been studied extensively, cf. the survey in [12] and there is an effective completeness criterion [10, 11]).

The degree of F is the maximum of the delays of the non-dummy (i.e. essential) variables. We say that $M \subseteq D$ is strongly complete in $N \subseteq D$ if $[M] = N$ (in [3, 4] for $d = 2$ the strong completeness in U is termed completeness in the first sense). Unless we need all delays this notion for $N = U$ or $N = D$ is too strong and therefore the following easy extensions of [4, Th. 3] are listed without their almost routine proofs.

Proposition 1. Let $M \subseteq U$ ($M \subseteq D$). Then M is strongly complete in $U(D)$ if and only if $\{f \in \Theta : (f, 0, \dots, 0) \in M\}$ is complete in Θ and M contains at least one operation of degree 1.

Subsets M of D satisfying $[M] = D$ are termed closed classes. Let L be the set of all closed classes ordered by inclusion. By its very definition L (being the subalgebra lattice of an algebra) is necessarily an algebraic lattice. We say that $M \subseteq \Theta$ is maximal (precomplete or coatom of L) in Θ if M is not complete in Θ while $M \cup \{f\}$ is complete in Θ for each $f \in \Theta \setminus M$. The structure of maximal classes is known, see e.g. [10 - 12]. We say that $M \subseteq D$ is strongly precomplete in $N \subseteq D$ if $M \subseteq N$, M is not strongly complete in N while $M \cup \{F\}$ is strongly complete for each $F \in N \setminus M$. For $X \subseteq \Theta$ set $A_X = \{F = \langle f, \delta_1, \dots, \delta_n \rangle \in D : f \in X \text{ for all } F \text{ of degree } 0\}$. $B = \{F \in D : \text{degree of } F \neq 1\}$. The following is easily proved from Prop 1 and [10, 11] and extends the last result from § 3 in [4].

Proposition 2. Each closed class distinct from D extends to a precomplete class. The precomplete classes consist of the classes A_M (M maximal in Θ) and the class B .

We could consider $M \subseteq D$ such that for each $f \in \Theta$ we have $(f, \delta_1, \dots, \delta_n) \in M$ for some delays $\delta_1, \dots, \delta_n \in \mathbb{N}$. Such a weak completeness of course coincides with the completeness in Θ , hence is of no particular interest. The following approach extends the one used in [4]. We say that $M \subseteq D$ is fully complete if for each $f \in \Theta$ there exists $\delta_f \in \mathbb{N}$ such that $(f, \delta, \dots, \delta) \in [M]$ for all $\delta > \delta_f$. This notion seems to be useful for it guarantees that any number of operations f_1, \dots, f_k can be synthesized with the common delay $\delta = \max(\delta_{f_1}, \dots, \delta_{f_k})$. Since in automata design seldom a single operation is needed, clearly such a simultaneous representability feature seems to be indispensable. Denote by e the unary identity operation defined by $e(x) = x$ for all $x \in d$ (i.e. (e, δ) represents the pure δ -delay). For $M \subseteq D$ let M^- denote the set of the $f \in \Theta$ that can be obtained via identification of variables from

operations $g \in \theta$ such that $(f, \delta_1, \dots, \delta_m) \in M$ for some $\delta_1, \dots, \delta_m \in \mathbb{N}$. We have the following extension of [7, Th. 2].

Proposition 3. Let $M \subseteq N \subseteq D$ and let $e \in N$. Then M is fully complete in N if and only if M^- is complete in N^- and there are relatively prime $x, y \in \mathbb{N}$ such that $(e, x) \in [M]$ and $(e, y) \in [M]$.

Proof: The necessity being obvious we prove the sufficiency only. We need the following fact sometimes attributed to Sylvester and used in this context in [7]. Let $x \in \mathbb{N}$ and $y \in \mathbb{N}$ be relatively prime and let $n = (x-1)(y-1)$. Then each $\delta \in \mathbb{N}$, $\delta \geq n$ can be uniquely expressed as the sum of x 's and y 's (i.e. $\delta = kx + ly$ for some $k, l \in \mathbb{N}$). Thus $(e, \delta) \in [M]$ for all $\delta \geq n$. Let $f \in N$. From M^- complete in N^- it follows that $(f, \delta_1, \dots, \delta_m) \in [M]$ for some $\delta_1, \dots, \delta_m \in \mathbb{N}$. Then for each $\delta > n + \max(\delta_1, \dots, \delta_m)$ the composition of f with $(e, \delta - \delta_1), \dots, (e, \delta - \delta_m) \in [M]$ yields the required result $(f, \delta, \dots, \delta) \in [M]$. ■

We say that M is fully precomplete in N if M is not fully complete in N but $M \cup \{F\}$ is for each $F \in N \setminus M$. We know the following fully precomplete classes. For $X \subseteq \theta$ set $X^+ = \{(f, \delta_1, \dots, \delta_m) \in D : f \in X, \delta_1, \dots, \delta_m \in \mathbb{N}\}$. For each prime p let E_p be the set of all $(f, \delta_1, \dots, \delta_n) \in D$ such that $\delta_i \equiv 0 \pmod{p}$ whenever the i -th variable is not dummy. We have:

Proposition 4. The set X^+ is fully precomplete if and only if X is maximal in θ . The set E is fully precomplete.

Proof: If X is not maximal in θ , then clearly X^+ is not fully precomplete. Let X be maximal in θ and let $F \in D \setminus X^+$. Then $(X \cup \{F\})^-$ is complete in θ , hence $X^+ \cup \{F\}$ is fully complete by Prop. 3 because $(e, 1) \in X^+$ and $(e, 2) \in X^+$.

Clearly E_p is not fully complete. Let $F = (f, \delta_1, \dots, \delta_n) \in D \setminus E_p$. Then there exists $1 \leq i \leq m$ such that the i -th variable is not dummy and $\delta_i \not\equiv 0 \pmod{p}$. The constant functions being available in E_p we can construct a unary nonconstant g such that $(g, \delta_1) \in H = [E_p \cup \{F\}]$. There exist unary $a, b \in \theta$ such that $m_i(x) = a(g(b(x)))$ satisfies $m_i(i) = 1$ and $m_i(x) = 0$ otherwise. From $(a, p) \in E_p$ and $(b, p) \in E_p$ we get $(m_i, \delta_i + 2p) \in H$. Further let $c \in \theta^{(d)}$ satisfy $c(1, 0, \dots, 0) = 0$, $c(0, 1, 0, \dots, 0) = 1, \dots, c(0, \dots, 0, 1) = d - 1$. In view of $(c, p) \in E_p$ and $c(m_0(x), \dots, m_{d-1}(x)) = x$ for all $x \in d$, we have $(e, \delta_i + 3p) \in H$. Here $\delta_i + 3p \equiv \delta_i \not\equiv 0 \pmod{p}$. Since clearly E_p^- is complete, by Prop. 3 we get H fully complete. ■

There are $M \subseteq D$ which belong to none of the above fully precomplete classes and yet are not fully complete (e.g. $\{(f, p, q)\}$ where f is Sheffer and p and q are distinct primes). Thus the characterization of full completeness by fully precomplete classes is an open problem (however we did not seriously try to solve it). Another type of completeness is introduced in [4]. We say that $M \subseteq D$ is uniformly complete in N if for each $f \in N^-$ the set $[M]$ contains $(f, \delta, \dots, \delta)$ for some $\delta \in \mathbb{N}$ (for $d = 2$ the papers [1, 3, 4] use the strong and uniform completeness in U only and call the latter the completeness in the second sense). The motivation other than esthetic for this definition is not quite clear although a plausible argument could perhaps be made based on synchronization and elimination of the worst hazards and races. As in [4] we have:

Proposition 5. A subset M of D is uniformly complete in D if and only if $(e, t) \in [M]$ and $(f_i, \delta_{i1}t, \dots, \delta_{in_i}t) \in [M]$ for a complete set $\{f_1, \dots, f_n\}$ and some $t \in \mathbb{N}$ and $\delta_{ij} \in \mathbb{N}$.

Proof: Necessity. Let $f \in \theta$ satisfy $f(0, 1, x) = x$ and $f(x, x, y) \equiv 1 + \max(x, y) \pmod{d}$ for all $x, y \in d$. Further let c_i denote the constant unary operation with value i . By assumption $(f, \delta, \delta, \delta) \in [M]$ for some $\delta \in \mathbb{N}$. Moreover $[M]$ contains $(c_i, 0)$ ($i = 0, 1$), hence the operation $h(x_1, x_2, x_3) = f(c_0(x_1), c_1(x_2), x_3)$ satisfies $(h, \delta, \delta, \delta) \in [M]$. We can identify all the variables and in view of $f(0, 1, x) = x$ ($x \in d$) obtain $(e, \delta) \in [M]$. Since $\{f\}$ is complete, the condition of the proposition is satisfied for $n = 1$, $f_1 = f$, $t = \delta$ and $\delta_{11} = \delta_{12} = \delta_{13} = 1$.

Sufficiency. Let $f \in \theta$. From $\{f_1, \dots, f_n\}$ complete we get $(f, \delta_1t, \dots, \delta_nt) \in [M]$ for some $\delta_1, \dots, \delta_n \in \mathbb{N}$. Set $\delta = \max(\delta_1, \dots, \delta_n)$ and $v_i = \delta - \delta_i$ ($i = 1, \dots, n$). If we replace the i -th argument of f by $e^{v_i}_{x_i}$ we get the required $(f, \delta t, \dots, \delta t) \in [M]$. ■

We turn our attention to U . Observe that U is not closed under composition (i.e. $[U] \not\subseteq U$). However, if the result of composition of functions from U belongs to U , then all inside functions have the same delays, i.e. to obtain $f(g_1(x_{11}), \dots, x_{1n_1}), \dots, g_m(x_{m1}, \dots, x_{mn_m}))$ all g_i 's must come from $G_i = (g_i, \delta, \dots, \delta)$ ($i = 1, \dots, m$).

The following proposition extends a result from [4, Th. 4].

Proposition 6. A subset M of U is uniformly complete in U if and only if $(e, t) \in [M]$ and $(f_i, \delta_{i1}t, \dots, \delta_{in_i}t) \in [M]$ ($i = 1, \dots, n$) for a complete set $\{f_1, \dots, f_n\}$ and some $t, \delta_1, \dots, \delta_n \in \mathbb{N}$.

Proof: Analogous to Prop. 2. ■

For $f \in \mathcal{O}$ let f^* be the unary operation defined by $f^*(x) = f(x, \dots, x)$ for all $x \in \underline{d}$ (the diagonal of f). As in [4, Th. 5] we have the following corollary.

Corollary 7. Let $F = (f, \delta_1, \dots, \delta_n)$. If $\{F\}$ is uniformly complete in U or D , then $\delta_1 = \dots = \delta_n = 0$.

Proof: Let $\{F\}$ be uniformly complete in U or D . It is easily seen that then $\delta_1 = \dots = \delta_n$. Suppose that their common value δ is positive. Clearly all the unary functions from $\{F\}$ are of the form $(f^*i, i\delta)$. However, the powers of f^* clearly do not exhaust all unary operations proving $\delta = 0$. ■

We say that $M \subseteq D$ is uniformly precomplete in $N \subseteq D$ if M is not uniformly complete while $M \cup \{F\}$ is uniformly complete for each $F \in N \setminus M$. We have:

Proposition 8. If M is maximal in \mathcal{O} , then M^+ and $M^+ \cap U$ are uniformly precomplete in D and U , respectively.

Proof: Since $M \neq \mathcal{O}$, clearly $M^+ \neq D$. Let $F = (f, \delta_1, \dots, \delta_n) \in D \setminus M^+$. Then $f \in \mathcal{O} \setminus M$ and $M \cup \{f\}$ is complete in \mathcal{O} . Moreover, $e \in M$, hence $(e, 1) \in M^+$ and the uniform completeness of $M^+ \cup \{F\}$ follows from Prop. 5. The proof of $M^+ \cap U$ precomplete in U is quite similar. ■ It is not clear whether there are uniformly precomplete classes in D other than M^+ , M maximal or whether each uniformly not complete subset of D extends to a uniformly precomplete one.

We turn our attention exclusively to the uniformly precomplete classes in U . Unfortunately the classes 7-9 from [4, paragraph preceding Th.6] are of no use for us (not even in the case $d = 2$) because our constants possess all possible delays and this modification, minor as it is, makes these classes already uniformly complete (in other words, our closure being stronger the closed classes are coarser). However the classes 10 and 11 [4, paragraph preceding Th. 6] inspire the following classes.

To simplify the notation write $\langle f, \delta \rangle$ instead of $(f, \delta, \dots, \delta)$. Let \leq be a partial order on \underline{d} . We say that $f \in \mathcal{O}$ is monotonic (antimonotonic) with respect to \leq if $f(x_1, \dots, x_n) \leq f(y_1, \dots, y_n)$

($f(x_1, \dots, x_n) \geq f(y_1, \dots, y_n)$) whenever $x_1 \leq y_1, \dots, x_n \leq y_n$. Let W_0 consist of $\langle f, \delta \rangle \in U$ such that f is monotonic (antimonotonic) whenever δ is even (odd). We have:

Lemma 9. Let \leq be a partial order with a least and greatest element. Then W_0 is uniformly precomplete in U .

Proof: First we prove $[W_0] \cap U = W_0$. Indeed let $F = \langle f, \delta' \rangle \in W_0$, $G_1 = \langle g_1, \delta'' \rangle \in W_0$ ($i=1, \dots, m$). The result of the simultaneous composition is

$H = \langle h, \delta' + \delta'' \rangle$ where $h = f(g_1, \dots, g_m)$. For $\tilde{x} \leq \tilde{y}$ we have $h(\tilde{x}) \leq h(\tilde{y})$ ($h(\tilde{x}) \geq h(\tilde{y})$) if $\delta' + \delta''$ is even (odd) proving $H \in W_0$. Moreover, the constant functions are exactly those both monotonic and antimonotonic and hence the constants with all possible delays appear in W_0 . Thus $[W_0] \cap U = W_0$.

Clearly there are functions neither monotonic nor antimonotonic and consequently W_0 is not uniformly complete.

Let $F = \langle f, \delta \rangle \in U \setminus W_0$. We must prove that $E = [W_0 \cup \{F\}] \cap U$ is uniformly complete in U . We show δ may be assumed to be even. Indeed suppose δ odd. Then there are $\tilde{z}, \tilde{t} \in \underline{d}^n$ such that $\tilde{z} \leq \tilde{t}$ while $f(\tilde{z}) \geq f(\tilde{t})$.

Let o and i be the least and greatest element of \leq . Define $h \in \mathcal{O}$ by setting $h(x) = i$ for $x \leq f(\tilde{z})$ and $h(x) = o$ otherwise. Now h is antimonotonic (if $h(x) \neq h(y)$ then $h(x) = o$ and $h(y) = i$, hence $x < f(\tilde{z})$ while $y < f(\tilde{z})$ proving $x \not\leq y$). Thus $(h, 1) \in W_0$ and $g = h(f)$ satisfies $g(\tilde{z}) = i \not\leq o = g(\tilde{t})$ and $\langle g, \delta+1 \rangle \in E \setminus W_0$ as required.

Now $\{g \in \mathcal{O} : \langle g, \delta \rangle \in E\}$ contains the set M of the monotonic functions and one non-monotonic function f . In view of the maximality of M in \mathcal{O} we have $M \cup \{f\}$ complete. Combining this with $(e, \delta) \in W_0$ and Prop. 6 we get E uniformly complete. ■

Following an idea from [4] we blow up the delays in W_0 by the same factor 2^v i.e. for each $v \in \mathbb{N}$ set

$$W_v = \{\langle f, 2^{v+1}q \rangle \in U : f \text{ monotonic}, q \in \mathbb{N}\}.$$

$$\cup \{\langle f, 2^v(2q+1) \rangle : f \text{ antimonotonic}, q \in \mathbb{N}\}$$

We have:

Proposition 10. Let \leq be a partial order with a least and a greatest element. Then for each $v \in \mathbb{N}$ the set W_v is uniformly precomplete in U .

Proof: The proof that $[W_v] \cap U = W_v$ is as in the proof of L 9. Let $F = \langle f, \delta \rangle \in U \setminus W_v$. To prove that $E = [W_v \cup \{F\}] \cap U$ is uniformly complete in U , we proceed as in the proof of L 9 with the following lemma.

Lemma 11. Let f be an n -ary operation which is either not monotonic or not antimonotonic. Then there are monotonic g_i ($i=1, \dots, n$) such that the n^2 -ary operation f_2 defined by $f_2(x_1, \dots, x_{n^2}) = f(g_1(x_1, \dots, x_n), \dots, g_n(x_{n^2-n+1}, \dots, x_{n^2}))$ is not antimonotonic.

Proof: Let f be not monotonic. Then there exists $\tilde{x} = (x_1, \dots, x_n) \in \underline{d}^n$ and $\tilde{y} = (y_1, \dots, y_n) \in \underline{d}^n$ such that $x_i \leq y_i$ ($i=1, \dots, n$) and $a = f(\tilde{x})$ and $b = f(\tilde{y})$ satisfy $a \not\leq b$. Let $g_i \in \mathcal{O}$ be defined by

$g_i(x) = x_i$ for $x \leq b$ and $g_i(x) = y_i$ otherwise ($i=1, \dots, n$). It is easy to check that all g_i are monotone. By this definition $f_2(\tilde{x}, \dots, \tilde{x}) = f(\tilde{y}) = b$ and $f(\tilde{y}, \dots, \tilde{y}) = f(\tilde{x}) = a$ proving that f_2 is antimonotonic. For f not antimonotonic the proof is similar with reversed inequalities and choosing $g_i(x) = y_i$ for $x > b$ and $g_i(x) = x_i$ otherwise).

We return to our proof. Using L 11 construct $\langle f_2, 2^{1+1}(2q+1) + 2^{r+1} \rangle \in E$ where f_2 is not antimonotonic. We repeat the same construction till we find $\langle h, 2^r(2q'+1) \rangle \in E$ such that h is not antimonotonic. From this point one can proceed as in the proof of L 11. ■

Let $p \leq d$ be a prime, let A be a p -element subset of \underline{d} and let s be a cyclic permutation of A . For the simplicity of notation we assume $A = \underline{p} = \{0, \dots, p-1\}$ and $s(x) = x \oplus 1$ for all $x \in \underline{p}$ where $x \oplus y$ is the element of \underline{p} congruent to $x+y$ mod p . For $f \in \mathcal{O}$ let f^* be the map $\underline{p} \rightarrow \underline{a}$ defined by $f^*(x) = f(x, \dots, x)$ for all $x \in \underline{p}$. We abbreviate $a \equiv b$ for $a \equiv b \pmod{p}$ and set $Z_0 = \{\langle f, \delta \rangle \in U : f^* = s^r \text{ and } \delta \equiv i \text{ for some } i \in \underline{p}\}$.

We have:

Proposition 12. The class Z_0 is uniformly precomplete in U .

Proof: First we prove $[Z_0] \cap U = Z_0$.
Indeed let $F = \langle f, \delta' \rangle \in Z_0$, $\delta' \equiv i'$ and $G_j = \langle g_j, \delta'' \rangle \in Z_0$, $\delta'' \equiv i''$ ($j=1, \dots, m$). The result of the simultaneous composition is $\langle h, \delta' + \delta'' \rangle$ where $h = f(g_1, \dots, g_m)$. Here clearly $h^*(x) = f(g_1^*(x), \dots, g_m^*(x)) = f(x \oplus i', \dots, x \oplus i'') = x \oplus i' \oplus i''$ for all $x \in \underline{p}$. Moreover $\delta' + \delta'' \equiv i' + i''$ proving $H \in Z_0$ and the claim. Since Z_0^- clearly is a proper subset of \mathcal{O} this shows that Z_0 is not uniformly complete in U .

Let $F = \langle f, \delta \rangle \in U \setminus Z_0$. We must prove that $E = [Z_0 \cup \{F\}] \cap U$ is uniformly complete in U . Let the unary operation $g \in \mathcal{O}$ be defined by $g(x) = f(x, \dots, x)$ for all $x \in \underline{d}$. Clearly $G = \langle g, \delta \rangle \in E$. Let $\delta \equiv i$. In view of $F \notin Z_0$ clearly $f^* \neq s^i$. Let δ' denote the least multiple of p not less than δ . First we prove that to each $x \in \underline{d}$ there exists $(f_x, \delta') \in E$ such that $f_x(x) \neq x$.

By definition Z_0 contains all $\langle h, p-i \rangle$ such that $h^* = s^{p-i}$, hence combining $\langle f, \delta \rangle$ with a suitable $\langle h, p-i \rangle$ we easily obtain f_x for $x \in \underline{d} \setminus \underline{p}$. Now let $x \in \underline{p}$. If $f(x) \oplus (p-i) \neq x$ we can construct f_x as well. Thus assume $f(x) = x \oplus i$. Similarly, if $f(x \oplus 1) \oplus (p-i-1) \neq x$, then using $\langle h_1, p-i-1 \rangle$ $h_1^* = s^{p-i-1}$ and $\langle h_2, 1 \rangle$, $h_2^* = s$ we can construct

$\langle f_x, \delta' \rangle$ where $f_x(y) = h_1(f(h_2(y)))$ for all $y \in \underline{d}$. Thus we assume $f(x \oplus 1) = x \oplus (i+1)$. Continuing in this fashion we get $f(x \oplus j) = x \oplus i \oplus j$ for all $j \in \underline{p}$. However, then $f(y) = y \oplus i$ for all $y \in \underline{d}$ contradicting $f \neq s^i$.

To complete the proof we use the following lemma which can be easily proved using the general completeness criterion [10,11]. We say that $f \in \mathcal{O}$ is homogeneous if $f(x, \dots, x) = x$ for all $x \in \underline{d}$.

Lemma 13. Let $M \subseteq \mathcal{O}$ contain all homogeneous operations. Then M is complete in \mathcal{O} if and only if for each $x \in \underline{d}$ M contains f_x such that $f_x(x, \dots, x) \neq x$.

By definition $\langle f, \delta' \rangle \in Z_0 \subseteq D$ for each homogeneous operation. The existence of $\langle f_x, \delta' \rangle$ ($x \in \underline{d}$) in D was shown above. Thus by the lemma there exists a set $M \subseteq \mathcal{O}$ complete in \mathcal{O} and such that $F = \{\langle f, \delta \rangle : f \in M\} \subseteq D$. To finish the proof it suffices to combine this with $(e, \delta') \in Z_0 \subseteq D$ and Prop. 6. ■
From Z_0 we can construct the class Z_v if we multiply all delays by p^v , i.e. set $Z_v = \{\langle f, \delta \rangle \in U : f^* = s^i \text{ and } \delta = p^v k, k \equiv i \text{ for some } i \in \underline{p}\}$.

Proposition 14. Each class Z_v is uniformly precomplete in U .

Proof: The proof is quite analogous to the proof of Prop. 5 with the following addition. Let

$F = \langle f, p^e k \rangle \in U \setminus Z_v$ where $e < v$ and $k \equiv i \neq 0$.

We show that then there is a unary $g \in \mathcal{O}$ and $s \in \{1\}$ such that $\langle g, p^v s \rangle \in D \setminus Z_v$.

Let $n = p^{v-1}$ and define h by $h(x) = f(x, \dots, x)$ for all $x \in \underline{d}$. Iterating n times the function

$H = \langle h, p^1 k \rangle$ we get $\langle h^n, p^v k \rangle \in D$.

Suppose $h^n \in Z_v$. Then $h^{n*} = s^i$, hence $h^{nj}(0)$ is the element of \underline{p} congruent to ij ($j=1, \dots, p$). By assumption $i \neq 0$ and therefore $0, h(0), \dots, h^{np-1}(0)$ is a cycle on which the elements of \underline{p} are evenly distributed in some order. It follows that

$h(x) \in \underline{d} \setminus \underline{p}$ for all $x \in \underline{p}$. Let the unary operation $m \in \mathcal{O}$ satisfy $m(x) = m(h(x)) = x$ for all $x \in \underline{p}$. By virtue of its definition $\langle m, p^{v+1} \rangle \in Z_v \subseteq D$. Let the unary operation

g be defined by $g(x) = m(h(x))$ for all $x \in \underline{d}$. Then $\langle g, p^v k + p^{v+1} \rangle$ is the required function (because $g^*(x) = x$ for all $x \in \underline{p}$ while the delay is of the form $p^v s$ where $s = k+p \equiv i$). Now the proof is virtually reduce to the one of Prop. 5. ■

It is far from clear whether the uniformly precomplete classes in U listed above exhaust all such classes. Moreover it is quite possible that there are $M \subseteq U$ which, although not uniformly complete, nevertheless belong in no uniformly precomplete

class. However it seems that for $d > 2$ the direct proof of M not contained in any of the above uniformly precomplete classes is already uniformly complete is probably too complex to contemplate.

Bibliography

- [1] L.A. BIRJUKOVA, V.B. KUDRJAVCEV, The completeness of functions with delays (Russian). Problemy Kibernet. 23 (1970) 5-27. MR 45 # 39. English translation: Systems theory research 23 (1973) 3-24.
- [2] R.E. KRÍČEVSKIJ, The realization of functions by superpositions (Russian). Problemy Kibernet. 2(1959) 123-138, German translation: Probleme der Kybernetik 2 (1963) 139-159.
- [3] V.B. KUDRJAVCEV, Completeness problems for a class of feedback free automata, Doklady AN SSSR 132 (1966) 272-274.
- [4] V.B. KUDRJAVCEV, A completeness theorem for a class of feedback-free automata (Russian). Problemy Kibernet. 8 (1962) 91-115. MR 30 # 28. German translation: Probleme der Kybernetik 8 (1965) 105-136.
- [5] V.B. KUDRJAVCEV, The power of sets of precomplete sets for certain functional systems connected with automata (Russian). Problemy Kibernet. No. 13 (1965) 45-47, MR 35 # 6493.
- [6] V.B. KUDRJAVCEV, On functional properties of logical nets (Russian). Math. Nachr. 55 (1973) 187-211.
- [7] H.H. LOOMIS Jr., Completeness of sets of delayed logic devices, IEEE Trans. on Electronic Computers EC-14 (1965) 157-174.
- [8] A.I. MAL'CEV, Iterative algebras and Post's varieties (Russian). Algebra i Logika (Sem.) 5 (1966) No. 2, pp. 5-24, MR 34 # 7424.
- [9] L. MARTIN, C. REISCHER, On the modular nets. Discrete Mathematics, 22 (1978).
- [10] I.G. ROSENBERG, Über die funktionale Vollständigkeit in dem mehrwertigen Logiken (Struktur der Funktionen von mehreren Veränderlichen auf endlichen Mengen). Rozprawy CS. Akademie Věd. Ser. Math. Nat.Sci., Vol. 80, 4 (1970), 3-93, MR 45 # 1732.
- [11] I.G. ROSENBERG, La structure des fonctions de plusieurs variables sur un ensemble fini, C.R. Acad. Sci. Paris, Ser. A.B. 260 (1965), 3817-19, MR 31 # 1185.
- [12] I.G. ROSENBERG, Completeness properties of multiple-valued logic algebras in Computer science and multiple-value logic (D. Rine ed.) North Holland 1976, pp. 149-185.

Added in proof: The existence of the following papers has been brought to our attention:

A. NOZAKI, Réalisation des fonctions définies dans un ensemble fini à l'aide des organes élémentaires d'entrée-sortie. Proc. Japan Acad. 46 (1970), 478-482.

T. HIKITA, A. NOZAKI, A completeness criterion for spectra, SIAM J. Comput. 6 (1977), 285-297.

T. HIKITA, A characterization for maximal incomplete sets of multivalued switching functions with delay. Preprint Faculty of Science, University of Tokio.

COMPLEX SPECTRAL LOGIC

Claudio Moraga

Universität Dortmund
Abteilung Informatik

Abstract

A spectral expansion of multiple-valued logical functions is studied. Chrestenson transforms and a mapping of logical functions into the unit circle in the complex plane are used. It is shown that a set of transformations in the function domain corresponds to permutations and/or complex scaling in the spectral domain. Spectral expansion of a function is unique and this is used to reduce complexity in multiple-valued logic design.

Introduction

Applications of the Rademacher-Walsh transforms to binary logic design are already known [1]. In a former paper [2] the author reported an extension of these methods to ternary logic design, unaware of the fact that in the meantime a good book on the subject was to be published [3].

If one compares the three above mentioned works, the following conclusions may be drawn:

i) In [2] an extension of the Rademacher - Walsh functions from binary to ternary was intended in the domain of integers. The resulting transformation matrix was however non orthogonal and the logic function could only be recovered up to certain symmetry classes after some spectral operations had been done. Notwithstanding, the proposed method provided a constructive means of applying to ternary logic design, most of the important results stated in [1] for the binary case.

ii) In [3] a formal mathematical extension of the Rademacher-Walsh functions to the prime p-valued case is presented, working on the unit circle in the complex plane [4,5,6]. These extended functions, the Chrestenson functions, form an orthogonal complete system. Interesting applications of spectral expansion to optimization of logic design are discussed.

iii) The method of spectral expansion presented in [3] does not allow to extend to the p-valued case all properties of spectral transformations as presented in [1]; particularly, does not cover the "disjoint spectral translation". It is however suggested that using *Characters*, other properties of interest for logic design might be obtained.

The present paper suggests one of these possible characters and shows that it preserves both the spectral transformations disclosed in [1,2], (including disjoint spectral translation) and refi-

nes the partition of the set of 2-place ternary functions into spectral families given in [2]. Moreover, it is shown that spectral expansion methods are applicable to any p-valued switching algebra and not only to the case of p a prime.

Notation and Definitions

D1: Let $V = \{0, 1, \dots, p-1\}$ be the set of values of a p-valued algebra. Let $Z \in V$ be a p-valued variable. \underline{Z} denotes an n-tuple $Z_{n-1} Z_{n-2} \dots Z_1 Z_0 \in V^n$.

Let z be a real number such that:

$$\sum_{i=0}^{n-1} Z_i p^i = z \quad (1)$$

It is obvious that for every n-tuple exists a unique z , but not for every z exists an n-tuple. When eq. (1) applies, it will be said that z is a *coding* of \underline{Z} and, conversely, that \underline{Z} is the *p-ary expansion* of z .

D2: $F: V^n \rightarrow V$ is a p-valued function which will be written $F(\underline{Z})$.

A *stepping* function $f(z)$ is a continuous representation of $F(\underline{Z})$, such that for every n-tuple \underline{Z}_i with coding z_i :

$$f(z_i + d) := F(\underline{Z}_i) \quad (2)$$

where $d \in [0, 1)$.

D3: [3] Any homomorphism of a group G into the multiplicative group of non-zero complex numbers is called a *character* of G .

D4: Let $u = \exp(2\pi\sqrt{-1}/p)$ and $h: (V, \oplus) \rightarrow (\mathbb{C}, \cdot)$ such that:

$$h(v) = u^v \quad (3)$$

where $v \in V$, \oplus stands for modulo p addition and \mathbb{C} is the set of complex numbers.

Let $v_0 = v_1 \oplus v_2$; then:

$$h(v_0) = u^{v_1 \oplus v_2} = u^{v_1} \cdot u^{v_2} = h(v_1) \cdot h(v_2)$$

It follows that h defines a character of (V, \oplus) .

D5: Chrestenson Functions, extended Rademacher-Walsh functions. [3,4,6]

Let w , ($0 \leq w \leq p^n - 1$), have p-ary expansion \underline{W} . The Chrestenson functions $X_w(z)$ are a set of

stepping functions defined in the interval $[0, p^n]$, by:

$$X_w(z) = u^{Ch(w,z)} \quad (4)$$

where

$$\begin{aligned} Ch(w,z) &= \sum_{s=0}^{n-1} W_{n-1-s} Z_s \\ &= \sum_{s=0}^{n-1} W_s Z_{n-1-s} \end{aligned} \quad (5)$$

Table 1 shows the Chrestenson functions for $p=3$, $n=2$, with $e_1 = 1/120^\circ$ and $e_2 = 1/240^\circ$.

D6: The notation \bar{Q} means the complement of Q , (i.e. $p-1-Q$) if Q is a p -valued variable or the complex conjugate value of Q if it is a complex quantity.

D7: Let J denote an $n \times n$ square matrix such that:

$$j_{r,s} := \text{if } (r=n-s+1) \text{ then } 1 \text{ else } 0$$

with $r, s \in \{1, \dots, n\}$.

Let P denote p^{n-1} ; let $*$ be an operation defined over V and let $t \in \{0, 1, \dots, P\}$.

The expression $(z * t)$ means the $*$ operation between the corresponding elements of the p -ary expansions of z and t respectively.

Let M be a square matrix. The expression tM , (Mt) , means a matrix product between the p -ary expansion of t considered as a row vector and the matrix M (the product between M and the p -ary expansion of t considered as a transposed row vector).

Similarly for $(M \# t)$ and $(z \# M)$, where $\#$ is some matrix operation.

Theorems

Th.1: [3,4] The set of Chrestenson functions is a complete orthogonal system:

$$\sum_{z=0}^P X_r(z) \cdot \overline{X_s(z)} := \text{if } r=s \text{ then } P \text{ else } 0 \quad (6)$$

If $g(z)$ is a stepping function representing a system of p -valued n -place logical functions such that for all w :

$$\sum_{z=0}^P g(z) \cdot \overline{X_w(z)} = 0$$

then $g(z) = 0$

Th.2: [3] $\forall w, z, t \in \{0, 1, \dots, P\}$

$$X_w(z) = X_z(w) \quad (7)$$

$$X_w(z \oplus t) = X_w(z) \cdot X_w(t) \quad (8)$$

Lemma 3: $X_w(z) = X_{wJ}(zJ)$ (9)

$$\text{and } X_w(z) = \overline{X_w[(p-1) \oplus z]} = \overline{X_{(p-1) \oplus w}(z)} \quad (10)$$

where \oplus stands for modulo p product.

Proof follows directly from Defs. 5, 6 and 7.

Th.4: Let $f(z)$ be a stepping function representing a p -valued n -place logical function, then:

$$S(w) = p^{-n} \sum_{z=0}^P hf(z) \overline{X_w(z)} \quad (11)$$

$$f(z) = h^{-1} \sum_{w=0}^P S(w) X_w(z) \quad (12)$$

where, from eq. (3), $h^{-1}(u^v) = v$.

$S(w)$ is called the *Spectrum* of the logical function. The following notation will often be used:

$$S(w) \leftrightarrow f(z); S(w) \leftrightarrow F(\underline{Z})$$

Proof: Replacing eq. (11) in eq. (12) and using Th.1 it follows that:

$$\begin{aligned} \sum_{w=0}^P S(w) X_w(z) &= \sum_{w=0}^P [p^{-n} \sum_{q=0}^P hf(q) \overline{X_w(q)}] X_w(z) = \\ &= p^{-n} \sum_{w=0}^P \sum_{q=0}^P hf(q) \overline{X_w(q)} X_w(z) = \end{aligned}$$

$$= p^{-n} \sum_{q=0}^P hf(q) \sum_{w=0}^P \overline{X_w(q)} X_w(z) = p^{-n} hf(z) p^n$$

Eq. (12) follows directly.

Lemma 5: $h(Z_k) = X_{p^{n-1-k}}(z)$ (13)

Proof: Let $w = p^{n-1-k}$. Then, in the p -ary expansion \underline{w} of w , $w_{n-1-k} = 1$ and all other digits of the expansion are 0. From eq. (5) it may be seen that the products in the summation which defines $Ch(w, z)$ will all be zero, except when $s = n-1-k$, where it takes the value Z_k . The lemma follows directly from this fact and eqs. (4) and (3).

Th. 6: Let $f(z)$ represent a function $F(\underline{Z})$ such that $f(0) = 0$.

$$\text{Let } f'(z) := \text{if } z \neq 0 \text{ then } f(z) \text{ else } v \in V \quad (14)$$

$$\text{Then } f(z) \leftrightarrow S(w) \implies$$

$$\implies f'(z) \leftrightarrow S(w) + p^{-n}(h(v)-1) \quad (15)$$

Proof: let $f'(z) \leftrightarrow S'(w)$

$$\begin{aligned} S(w) &= p^{-n} \sum_{z=0}^P hf(z) \overline{X_w(z)} = \\ &= p^{-n} \left[\sum_{z=1}^P hf(z) \overline{X_w(z)} + hf(0) \overline{X_w(0)} \right] = \\ &= p^{-n} \left[\sum_{z=1}^P hf(z) \overline{X_w(z)} + 1 \right] \end{aligned}$$

Similarly,

$$S'(w) = p^{-n} \left[\sum_{z=1}^P hf(z) \overline{X_w(z)} + h(v) \right]$$

$$\text{then } S'(w) = S(w) + p^{-n}(h(v) - 1)$$

Th.7: Argument translation [3].

$$f(z) \leftrightarrow S(w) \implies f(z \oplus a) \leftrightarrow X_w(a) S(w) \quad (16)$$

where $a \in (0, 1, \dots, P)$.

Th.8: (Extended from [3]). Linear transformation of the arguments.

$$f(z) \leftrightarrow S(w) \Rightarrow f(L \otimes z) \leftrightarrow S(w \otimes JL^{-1}J) \quad (17)$$

L is a non singular n -square matrix whose elements belong to V if p is a prime, or belong to the subset of V which contains no zero divisors if p is not a prime. This restriction guarantees that the only effect considered is that of a permutation on $f(z)$ without loss of information.

Th.9: Disjoint Spectral Translation.

$$f(z) \leftrightarrow S(w) \Rightarrow f(z) \otimes Z_k \leftrightarrow S(w \otimes p^{n-1-k}) \quad (18)$$

where \otimes stands for modulo p substraction.

Proof: Let $f(z) \otimes Z_k \leftrightarrow S'(w)$

$$\begin{aligned} S'(w) &= p^{-n} \sum_{z=0}^P h(f(z) \otimes Z_k) \overline{X_w(z)} \\ &= p^{-n} \sum_{z=0}^P hf(z)h(Z_k) \overline{X_w(z)} \\ &= p^{-n} \sum_{z=0}^P hf(z) X_{p^{n-1-k}} \overline{X_w(z)} \\ &= p^{-n} \sum_{z=0}^P hf(z) \overline{X_{(w \otimes p^{n-1-k})}(z)} \\ &= S(w \otimes p^{n-1-k}) \end{aligned}$$

It should be noticed that disjoint spectral translation on $f(z)$ produces only a permutation on $S(w)$. Following the steps of the proof it may be seen that this is achieved because of the chosen character. As shown in lemma 5, the image of a logical variable in the unit circle is in this case a Chrestenson function.

Lemma 10: Permutation on the value of $f(z)$.

$$f(z) \leftrightarrow S(w) \Rightarrow f(z) \otimes a \leftrightarrow h(a)S(w) \quad (19)$$

where $a \in V$.

Proof: Let $f(z) \otimes a \leftrightarrow S'(w)$

$$\begin{aligned} S'(w) &= p^{-n} \sum_{z=0}^P h(f(z) \otimes a) \overline{X_w(z)} \\ &= p^{-n} \sum_{z=0}^P h(a)hf(z) \overline{X_w(z)} = h(a)S(w) \end{aligned}$$

Th.11: Complementation of $f(z)$.

$$f(z) \leftrightarrow S(w) \Rightarrow \overline{f(z)} \leftrightarrow h(-1)S[(p-1) \otimes w] \quad (20)$$

Proof: Let $\overline{f(z)} \leftrightarrow S'(w)$

$$\begin{aligned} S'(w) &= p^{-n} \sum_{z=0}^P h(p-1-f(z)) \overline{X_w(z)} \\ &= p^{-n} \sum_{z=0}^P h(-1)h[-f(z)] \overline{X_w(z)} \end{aligned}$$

$$\begin{aligned} &= h(-1)p^{-n} \sum_{z=0}^P hf(z) \overline{X_{(p-1) \otimes w}(z)} \\ &= h(-1)S[(p-1) \otimes w] \end{aligned}$$

Th.12: (Adapted from [3])

$$X_w(L \otimes z) = X_{(w \otimes JLJ)}(z) \quad (21)$$

Proof: Let $[a]_i$ denote A_i from the p -ary expansion of a .

$$\text{Let } g = \sum_{s=0}^{n-1} W_s [L \otimes z]_{n-1-s}$$

Then, after Defs. 4 and 5, $u^g = X_w(L \otimes z)$.

$$\text{But } [L \otimes z]_{n-1-s} = \sum_{q=0}^{n-1} L_{s,q} Z_{n-1-q}$$

$$\begin{aligned} \text{Then } g &= \sum_{s=0}^{n-1} W_s \sum_{q=0}^{n-1} L_{s,q} Z_{n-1-q} = \\ &= \sum_{q=0}^{n-1} Z_{n-1-q} \sum_{s=0}^{n-1} W_s L_{s,q} = \sum_{q=0}^{n-1} Z_{n-1-q} [w \otimes JL]_{n-1-q} = \\ &= \sum_{q=0}^{n-1} Z_{n-1-q} [w \otimes JLJ]_q \end{aligned}$$

eq. (21) follows directly.

Th.13: Let $f'(z)$ represent $F(Z_{n-1} \dots \bar{Z}_k \dots Z_1 Z_0)$

Then $f(z) \leftrightarrow S(w) \Rightarrow$

$$\Rightarrow f'(z) \leftrightarrow X_{(w \otimes JL_k J)}(p^k) S(w \otimes JL_k J) \quad (22)$$

Proof: $\bar{Z}_k = (p-1) \otimes (1 \otimes Z_k) = (p-1) \otimes Z_k \otimes (p-1)$

$$\begin{aligned} \text{then } f'(z) &= f(z \otimes (p-2) \otimes Z_k p^k \otimes p^k) \\ &= f(z \otimes 2 \otimes Z_k \otimes p^k \otimes p^k) \end{aligned}$$

Let L_k be an n -square matrix with elements l_{ij} such that:

$$l_{ij} := \begin{cases} \text{if } (i=j=n-1-k) \text{ then } (p-1) \text{ else} \\ \text{if } (i=j \neq n-1-k) \text{ then } 1 \text{ else } 0 \end{cases}$$

It follows that:

$$L_k = L_k^{-1} \quad \text{and} \quad f'(z) = f(L_k \otimes z \otimes p^k)$$

Let $f'(z) \leftrightarrow S'(w)$

$$\text{then } S'(w) = p^{-n} \sum_{z=0}^P hf(L_k \otimes z \otimes p^k) \overline{X_w(z)} =$$

$$= p^{-n} \sum_{z=0}^P hf(L_k \otimes z \otimes p^k) \overline{X_{(L_k^{-1} \otimes L_k \otimes z)}(z)} =$$

$$= p^{-n} \sum_{z=0}^P hf(L_k \otimes z \otimes p^k) \overline{X_{(w \otimes JL_k J)}(L_k \otimes z)}$$

(This last step is an application of Th. 12)

Eq. (22) follows directly from Th. 7 and Lemma 3.

Lemma 14: $F(\underline{Z}) \leftrightarrow S(w) \Rightarrow$

$$\Rightarrow F(\underline{Z}) \leftrightarrow X_w^{(P/2)} S[(p-1) \otimes w] \quad (23)$$

Proof: Let $f'(z)$ represent $F(\underline{Z})$. Then,

$$f'(z) = f(z) \otimes_{k=0}^{n-1} [(p-2) \otimes Z_k \otimes p^k \otimes p^k]$$

From Th. 13 now $L_k = -I_n$ (n -square unity matrix), $L_k = L_k^{-1}$ and $(w \otimes J L_k J) = (p-1) \otimes w$. It follows that:

$$f'(z) \leftrightarrow X_{(p-1) \otimes w}^{(P/2)} S[(p-1) \otimes w]$$

Eq. (23) is obtained after Lemma 3.

Lemma 15: $F(\underline{Z}) \leftrightarrow S(w) \Rightarrow$

$$\Rightarrow \bar{F}(\underline{Z}) \leftrightarrow h(-1) X_w^{(P/2)} \bar{S}(w) \quad (24)$$

Proof follows directly from Lemma 14 and Th. 11.

Symmetric representation of odd-valued functions

D8: Let $V_s = \{-\lfloor p/2 \rfloor, \dots, -1, 0, 1, \dots, \lfloor p/2 \rfloor\}$

(p odd).

$Y \in V_s$ is a symmetric odd-valued variable.

\underline{Y} is an n -tuple and y is its coding, i.e.,

$$y = \sum_{i=0}^{n-1} Y_i p^i$$

\underline{Y} is the p -ary expansion of y and $f(y)$ is a stepping function representing $F(\underline{Y})$, where now

$$F: V_s^n \rightarrow V_s.$$

D9: Symmetric Chrestenson Functions.

For w and y such that $-\lfloor p/2 \rfloor \leq w, y \leq \lfloor p/2 \rfloor$, where w and y have p -ary expansions $\underline{W}, \underline{Y} \in V_s^n$ respectively, define:

$$Ch(w, y) = \sum_{q=0}^{n-1} W_q Y_{n-1-q} = \sum_{q=0}^{n-1} W_{n-1-q} Y_q \quad (25)$$

$$X_w(y) = u^{Ch(w, y)}$$

It is easy to see that under multiplication, these symmetric Chrestenson functions are isomorphic to those in D5. They also form a complete orthogonal system.

Considering that for the symmetric odd-valued algebra operations \oplus and \otimes continue to be modulo p addition and product respectively, only that now the corresponding residues have values from V_s , it is simple to show that all previously stated theorems, except for Th. 11 and Th. 13 may be directly extended to the symmetric case without other change than the range of the subscripts and residues.

Because of the fact that in the symmetric case $\bar{Y} = -Y$, it is easy to show that:

$$X_w(-y) = \bar{X}_w(y) \text{ and } X_{-w}(y) = \bar{X}_w(y) \quad (26)$$

which will be used in the following theorem:

Th.17: $f(y) \leftrightarrow S(w) \Rightarrow \bar{f}(y) \leftrightarrow \bar{S}(-w) \quad (27)$

Proof: $\bar{f}(y) = -f(y)$; $h\bar{f}(y) = h(-f(y)) = h\bar{f}(y)$

Let $\bar{f}(y) \leftrightarrow S'(w)$ and let $\beta = \lfloor p/2 \rfloor$; then:

$$S'(w) = p^{-n} \sum_{y=-\beta}^{\beta} h\bar{f}(y) \bar{X}_w(y)$$

$$\bar{S}(w) = p^{-n} \sum_{y=-\beta}^{\beta} h\bar{f}(y) \bar{X}_{-w}(y) = S(-w)$$

$$S'(w) = \bar{S}(-w)$$

Comparing eqs. (27) and (20) it may be seen that for the symmetric case, complementation of the logical function produces only a mirror permutation and complex conjugation of the spectral elements without complex scaling as required in the non symmetric case.

Moreover, in the symmetric case, complementation of arguments may be handled as linear transformation of the arguments (Th. 8), by assigning -1 entries to those places of the main diagonal of L which correspond to the variables to be complemented, 1 entries to the remaining elements of the main diagonal and 0 elsewhere. If all arguments are complemented, $L = -I$ and $JLJ = -I$, so that:

$$f(-y) \leftrightarrow S(-w) \quad (28)$$

and then,

$$\bar{f}(-y) \leftrightarrow \bar{S}(w) \quad (29)$$

which proves the following

Corollary: Self dual functions have a real Spectrum.

Classification of 2-place ternary functions and Multiple-valued Logic Design.

By full induction it has been possible to prove that using all previous theorems, except Th. 7 and Th. 10, the same partition of the 19,683 2-place ternary functions in 25 (75) families as presented in [2] is obtained, but adding uniqueness to the relation between a logical function and its spectrum. It is also shown that without using Th. 6 but including Th. 7 and Th. 10, a partition in only 12 classes is obtained. 10 from these classes contain one or more threshold functions and the other two classes, which comprise only 243 functions, contain one or more functions with simple Bilinear Separability [7].

From the logic design point of view, spectral logic may be considered as an alternative for introducing diversity on a very small set of connectives, reduce diversity on a very large set of connectives, but, more important, as an alternative for reducing complexity of realizations. In [3] very interesting autocorrelation techniques are presented for the optimization of binary and multiple-valued logic design. The author would rather take here a different approach, because he feels that still very little is known on complexity of realizations of multiple-valued digital systems. Moreover, using autocorrelation techniques for optimization of multiple-valued logic design would require restricting p to be a prime in order that

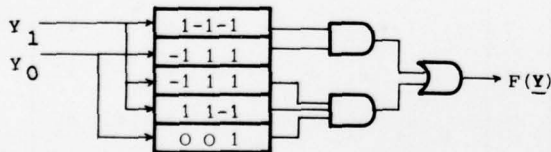
(V, \oplus, \otimes) be a field. At the present stage of development in multiple-valued logic design, setting p to be a prime seems to be too severe a restriction: it would rule out 4 and 8 valued logics, which exhibit a high degree of compatibility with binary circuitry, as well as 10-valued logic, which might lead to the realization of decimal machines. On the other hand optimization using correlation with linear functions has been investigated with encouraging results [8,9] and optimization by heuristic computer search seems still to be adequate. More studies on complexity of realization and functional decomposition of multiple-valued functions are indeed needed in order to make better use of spectral logic possibilities.

The rather limited experience with multiple-valued threshold functions allow however to claim that, in general, they provide low complexity realizations. The earlier mentioned partition of the 2-place ternary functions proves that almost all of them are realizable by a threshold "core" and spectral transformations. For binary systems it has been shown [1] that such is the case at least up to 4-place functions. This gives hopes that for higher bases and larger argument dimensions a similar situation might be found.

Example: Let $F(Y)$ be a 2-place ternary function in symmetric notation. Let $F(Y)$ be described by the row vector $[f(-4), f(-3), \dots, f(0), \dots, f(3), f(4)] = [-1, 1, 1, 0, 0, 1, -1, -1, -1]$. A simple realization for $F(Y)$ is looked for.

a) direct realization using as basic connectives MIN, MAX and all monotonic unary functions.

Following [10] the logic design whose diagram is shown below may be obtained, where the rectangular boxes represent gates which realize unary functions and their label indicates the corresponding output when the input is respectively -1, 0 or 1.



b) realization by spectral logic.

Let the spectral elements be coded by a three digits number $n_0 n_1 n_2$, meaning that the complex value for that element is $(n_0 + n_1/120^\circ + n_2/240^\circ)$ divided by 9. Introducing Def. 8 and Def. 9 in Th. 4 and summing over y from -4 to 4, the following is obtained:

$F(Y) \leftrightarrow (201 \ -101 \ 204 \ 720 \ \underline{-101} \ 120 \ 201 \ 420 \ 120)$
(where $S(0)$ is underlined).

Assume that a simple realization on a threshold core is desired. From Table 2, which is a short summary of the 12 classes partition, it is found that there exists a threshold function $H(Y)$ whose spectrum is a permutation of that from $F(Y)$:

$H(Y) \leftrightarrow (201 \ -101 \ 204 \ -101 \ \underline{120} \ 720 \ 120 \ 201 \ 420)$

To modify $S(0)$ a disjoint spectral translation is required (Th.9). It may easily be shown that:

$$H(Y) \oplus Y_1 \leftrightarrow$$

$$\leftrightarrow (204 \ 201 \ -101 \ 720 \ \underline{-101} \ 120 \ 420 \ 120 \ 201)$$

To permute other spectral elements without changing $S(0)$, a linear transformation of the argument is needed (Th.8). It may be shown that if:

$$Y_0 \rightarrow Y_0 \oplus Y_1 \quad \text{i.e.} \quad L = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad (\text{See Table 3})$$

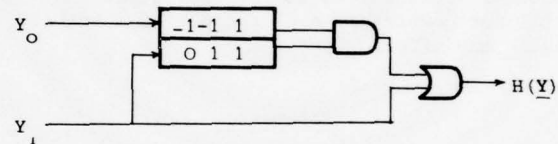
$$H(Y_1, Y_0 \oplus Y_1) \oplus Y_1 \leftrightarrow$$

$$\leftrightarrow (201 \ -101 \ 204 \ 720 \ \underline{-101} \ 120 \ 201 \ 420 \ 120)$$

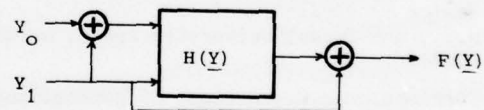
which is the required spectrum. It follows that:

$$F(Y) = H(Y_1, Y_0 \oplus Y_1) \oplus Y_1$$

From Table 2, $H(Y) = [-1, -1, 0, 0, 0, 1, 1, 1, 1]$, which using the same type of core gates as for $F(Y)$ in a), has the following realization:



and, of course, also a direct threshold realization with weights (1, 2) and thresholds (0.5, -1.5). This leads to the following final realization of $F(Y)$:



Spectral logic design assumes that complexity criteria for multiple-valued hardware are known and that algorithms for finding low complexity functions are available, which at the present seems to be an open problem with some empirical answers. In the example above discussed, threshold core realizability was chosen to be an acceptable low complexity criterion; moreover, means of recognizing ternary threshold functions do exist [11]. Spectral logic design also assumes that there is a simple and reliable realization both for the core gate(s) and the modulo adders, which for the case of a threshold core as well as for the modulo adders seems to be a reasonable assumption in terms of I²L technology. Finally, spectral logic design methods lend themselves usually more to computational work rather than to hand-work, specially when recognizability of some spectra belonging to a same spectral class becomes difficult as after argument translation (Th.7) or complementation of some arguments in the non symmetric case (Th.13).

Concluding remarks and future work

Spectral logic seems to be an attractive alternative for multiple-valued logic design. It is not restricted to use only the Chrestenson functions: any set of functions which constitute a complete orthogonal system will allow a spectral expansion of multiple-valued logical functions and might well have quite different properties than those discussed in this paper. Haar functions [12] for instance, are studied in [3] and it is shown that they might well be used to analyse local perturbations in logical functions. In this paper, spectral expansion of a character of a multiple-valued logical function by means of the Chrestenson functions is shown to have interesting properties for low complexity multiple-valued logic design.

As stated before, more studies on complexity of multiple-valued logical functions and functional decomposition seem to be needed in order to be able to use in full extend the possibilities that the still not completely explored set of spectral logics may offer.

References

- [1] Edwards C.R.: The application of the Rademacher-Walsh transform to boolean classification and threshold logic synthesis. *Trans. IEEE* C-24, 48-62, (1975)
- [2] Moraga C.: Ternary Spectral Logic. *Proceedings 7th ISMVL*, Charlotte N.C., U.S.A., 7-12, (1977)
- [3] Karpovsky M.: "Finite Orthogonal Series in the design of digital devices", John Wiley & Sons, N.Y., and Israel University Press, Jerusalem, (1976)
- [4] Chrestenson H.E.: A class of generalized Walsh functions, *Pacific J. Math.* 5, 17-31, (1955)
- [5] Selfridge R.E.: Generalized Walsh transforms, *Pacific J. Math.*, 5, 451-480, (1955)
- [6] Muzio J.: Concerning Transforms for three-valued systems, *Report CS 77001-R*, Virginia Polytechnic Institute and State University, (1977)
- [7] Nazarala J. and Moraga C.: Bilineal Separability of ternary functions. *Proceedings 5th ISMVL*, Bloomington, Indiana, USA, 305-315, (1975)
- [8] Moraga C.: Spectral Logic Design, *Bericht 57*, Abteilung Informatik, Universität Dortmund, (1978)
- [9] Moraga C.: Introducing disjoint spectral translation in multiple-valued spectral logic design, *Electronics Letters*, (to be published) (1978)
- [10] Moraga C.: Mehrwertige Schaltalgebra, Class Notes WS74/75, Abteilung Informatik, Universität Dortmund, (1975). (In german)
- [11] Nazarala J. and Moraga C.: Minimal realization of ternary threshold functions, *Proceedings*

4th ISMVL, Morgantown, W.Va., USA, 347-358, (1974)

- [12] Haar A.: Zur Theorie der orthogonalen Funktionensysteme, *Math. Ann.*, 69, 331-371, (1910)

Table 1: Chrestenson functions for p=3 and n=2

w \ z	0	1	2	3	4	5	6	7	8
0	1	1	1	1	1	1	1	1	1
1	1	1	1	e ₁	e ₁	e ₁	e ₂	e ₂	e ₂
2	1	1	1	e ₂	e ₂	e ₂	e ₁	e ₁	e ₁
3	1	e ₁	e ₂	1	e ₁	e ₂	1	e ₁	e ₂
4	1	e ₁	e ₂	e ₁	e ₂	1	e ₂	1	e ₁
5	1	e ₁	e ₂	e ₂	1	e ₁	e ₁	e ₂	1
6	1	e ₂	e ₁	1	e ₂	e ₁	1	e ₂	e ₁
7	1	e ₂	e ₁	e ₁	1	e ₂	e ₂	e ₁	1
8	1	e ₂	e ₁	e ₂	e ₁	1	e ₁	1	e ₂
$X_w(z)$									

w \ y	$\bar{4}$	$\bar{3}$	$\bar{2}$	$\bar{1}$	0	1	2	3	4
-4	e ₂	e ₁	1	e ₁	1	e ₂	1	e ₂	e ₁
-3	e ₁	1	e ₂	e ₁	1	e ₂	e ₁	1	e ₂
-2	1	e ₂	e ₁	e ₁	1	e ₂	e ₂	e ₁	1
-1	e ₁	e ₁	e ₁	1	1	1	e ₂	e ₂	e ₂
0	1	1	1	1	1	1	1	1	1
1	e ₂	e ₂	e ₂	1	1	1	e ₁	e ₁	e ₁
2	1	e ₁	e ₂	e ₂	1	e ₁	e ₁	e ₂	1
3	e ₂	1	e ₁	e ₂	1	e ₁	e ₂	1	e ₁
4	e ₁	e ₂	1	e ₂	1	e ₁	1	e ₁	e ₂
$X_w(y)$									

Table 2: Summary of the partition of the set of 2-place ternary functions.
 Table lists representative threshold functions in each class and their corresponding spectra. Spectral elements are given as a three digits number $n_0 n_1 n_2$ meaning that the value of that element is $(n_0 + n_1/120^0 + n_2/240^0)/9$

Class	Function f(y)			Spectrum S(w)									
	y			w									
	-4 -3 -2	-1 0 1	2 3 4	-4	-3	-2	-1	0	1	2	3	4	
1	-1-1-1	-1-1-1	-1-1 0	102	-110	210	-110	108	102	210	102	-110	
	-1-1-1	-1-1-1	-1-1 1	201	-101	120	-101	-107	201	120	201	-101	
	0 0 0	0 0 0	0 0 1	210	102	-110	102	810	210	-110	210	102	
	-1-1-1	0 0 1	1 1 1	102	102	102	-110	-110	810	210	210	210	
2	-1-1-1	-1-1-1	-1 0 0	-101	120	201	-220	207	204	120	201	-101	
	-1-1-1	-1-1-1	-1 1 1	102	-110	210	-202	-205	402	-110	210	102	
	0 0 0	0 0 0	0 1 1	201	-101	120	204	720	420	-101	120	201	
	-1-1-1	0 0 0	0 0 1	210	102	-110	240	402	702	-110	210	102	
	-1-1-1	0 1 1	1 1 1	-101	-101	-101	-220	-220	720	120	120	120	
	-1-1 0	0 0 1	1 1 1	201	-101	204	-101	120	720	120	201	420	
3	-1-1-1	-1-1-1	-1 0 1	000	000	000	-300	006	303	030	300	003	
	-1-1-1	-1 0 1	1 1 1	000	000	000	-300	-300	600	300	300	300	
4	-1-1-1	-1-1-1	0 0 0	000	000	000	-330	306	306	000	000	000	
	-1-1-1	0 0 0	0 0 0	000	000	000	360	603	603	000	000	000	
	-1-1-1	1 1 1	1 1 1	000	000	000	-330	-330	630	000	000	000	
5	-1-1-1	-1-1-1	0 0 1	210	102	-110	-410	105	405	-110	210	102	
	-1-1-1	-1 0 0	0 0 0	120	120	120	150	504	504	201	201	201	
	-1-1-1	-1-1-1	0 1 1	201	-101	120	-401	-104	504	-101	120	201	
	-1-1-1	-1-1 0	0 0 0	-110	-110	-110	-140	405	405	102	102	102	
	-1-1-1	-1 1 1	1 1 1	-110	-110	-110	-410	-410	510	210	210	210	
6	-1-1-1	-1-1 0	-1 0 0	-330	030	000	030	306	303	000	303	003	
	-1-1-1	-1-1 1	-1 1 1	003	-300	000	-300	-303	300	000	300	303	
	-1-1-1	-1 0 0	0 0 1	330	000	030	030	303	603	000	300	303	
	-1-1-1	-1-1 0	0 1 1	000	-300	030	-300	003	603	003	000	303	
7 *	-1-1-1	-1-1 0	-1 0 1	-110	-110	-110	-110	105	402	-110	402	105	
	-1-1 1	0 1 1	1 1 1	-110	-202	102	-110	-140	510	-110	210	240	
8	-1-1-1	-1-1 0	0 0 1	120	-101	-220	-220	204	504	-101	201	204	
	-1-1-1	-1-1 1	0 1 1	102	-202	-110	-410	-202	402	102	210	402	
9	-1-1-1	0 0 0	1 1 1	000	000	000	000	000	900	000	000	000	
10	-1-1 0	-1 0 0	0 0 1	540	-110	-110	-110	402	402	-110	402	402	
11 **	-1 0 0	-1 0 0	0 1 1	300	030	030	003	300	300	003	300	300	
12 **	-1-1 0	-1-1 0	0 0 1	330	-300	-300	-300	303	303	-300	303	303	

* Class contains also functions MAX and MIN

** Listed function is bilinear separable

Table 3: Linear transformation of the argument for $p=3, n=2$, symmetric. (Th.8).
 (Matrix L and its inverse are listed as a one dimensional array).
 Disjoint Spectral Translation for $p=3, n=2$, symmetric. (Th.9).

L	$L \otimes y$	$w \otimes JL^{-1}J$	L^{-1}
1 0 0 1	-4 -3 -2 -1 0 1 2 3 4	-4 -3 -2 -1 0 1 2 3 4	1 0 0 1
-1-1-1 0	-2 4 1 3 0 -3 -1 -4 2	1 -2 4 3 0 -3 -4 2 -1	0-1-1 1
-1-1-1 1	-3 4 -1 2 0 -2 1 -4 3	1 2 -3 -4 0 4 3 -2 -1	1 1 1-1
-1-1 0-1	-2 3 -1 4 0 -4 1 -3 2	1 3 -4 -2 0 2 4 -3 -1	-1 1 0-1
-1-1 0 1	-4 3 1 2 0 -2 -1 -3 4	1 -3 2 4 0 -4 -2 3 -1	-1-1 0 1
-1-1 1-1	-3 2 1 4 0 -4 -1 -2 3	1 -4 3 2 0 -2 -3 4 -1	1-1 1 1
-1-1 1 0	-4 2 -1 3 0 -3 1 -2 4	1 4 -2 -3 0 3 2 -4 -1	0 1-1-1
-1 0-1-1	2 4 3 1 0 -1 -3 -4 -2	3 2 4 1 0 -1 -4 -2 -3	-1 0 1-1
-1 0-1 1	3 2 4 -1 0 1 -2 -4 -3	-4 -2 -3 1 0 -1 3 2 4	-1 0-1 1
-1 0 0-1	4 3 2 1 0 -1 -2 -3 -4	4 3 2 1 0 -1 -2 -3 -4	-1 0 0-1
-1 0 0 1	2 3 4 -1 0 1 -4 -3 -2	-2 -3 -4 1 0 -1 4 3 2	-1 0 0 1
-1 0 1-1	3 2 4 1 0 -1 -4 -2 -3	2 4 3 1 0 -1 -3 -4 -2	-1 0-1-1
-1 0 1 1	4 2 3 -1 0 1 -3 -2 -4	-3 -4 -2 1 0 -1 2 4 3	-1 0 1 1
-1 1-1-1	-1 4 -3 -2 0 2 3 -4 1	3 -2 -1 -4 0 4 1 2 -3	1 1-1 1
-1 1-1 0	1 4 -2 -3 0 3 2 -4 -1	-4 2 -1 3 0 -3 1 -2 4	0-1 1-1
-1 1 0-1	1 3 -4 -2 0 2 4 -3 -1	-2 3 -1 4 0 -4 1 -3 2	-1-1 0-1
-1 1 0 1	-1 3 -2 -4 0 4 2 -3 1	4 -3 -1 -2 0 2 1 3 -4	-1 1 0 1
-1 1 1 0	-1 2 -4 -3 0 3 4 -2 1	2 -4 -1 -3 0 3 1 4 -2	0 1 1 1
-1 1 1 1	1 2 -3 -4 0 4 3 -2 -1	-3 4 -1 2 0 -2 1 -4 3	1-1-1-1
0-1-1-1	2 1 -3 4 0 -4 3 -1 -2	3 1 -4 2 0 -2 4 -1 -3	1-1-1 0
0-1-1 0	4 1 -2 3 0 -3 2 -1 -4	4 1 -2 3 0 -3 2 -1 -4	0-1-1 0
0-1-1 1	3 1 -4 2 0 -2 4 -1 -3	2 1 -3 4 0 -4 3 -1 -2	-1-1-1 0
0-1 1-1	3 -1 -2 4 0 -4 2 1 -3	-4 1 3 -2 0 2 -3 -1 4	-1 1-1 0
0-1 1 0	2 -1 -4 3 0 -3 4 1 -2	-2 1 4 -3 0 3 -4 -1 2	0 1-1 0
0-1 1 1	4 -1 -3 2 0 -2 3 1 -4	-3 1 2 -4 0 4 -2 -1 3	1 1-1 0
0 1-1-1	-4 1 3 -2 0 2 -3 -1 4	3 -1 -2 4 0 -4 2 1 -3	-1-1 1 0
0 1-1 0	-2 1 4 -3 0 3 -4 -1 2	2 -1 -4 3 0 -3 4 1 -2	0-1 1 0
0 1-1 1	-3 1 2 -4 0 4 -2 -1 3	4 -1 -3 2 0 -2 3 1 -4	1-1 1 0
0 1 1-1	-3 -1 4 -2 0 2 -4 1 3	-2 -1 3 -4 0 4 -3 1 2	1 1 1 0
0 1 1 0	-4 -1 2 -3 0 3 -2 1 4	-4 -1 2 -3 0 3 -2 1 4	0 1 1 0
0 1 1 1	-2 -1 3 -4 0 4 -3 1 2	-3 -1 4 -2 0 2 -4 1 3	-1 1 1 0
1-1-1-1	-1 -2 3 4 0 -4 -3 2 1	3 -4 1 -2 0 2 -1 4 -3	-1 1 1 1
1-1-1 0	1 -2 4 3 0 -3 -4 2 -1	-2 4 1 3 0 -3 -1 -4 2	0-1-1-1
1-1 0-1	1 -3 2 4 0 -4 -2 3 -1	-4 3 1 2 0 -2 -1 -3 4	1-1 0-1
1-1 0 1	-1 -3 4 2 0 -2 -4 3 1	2 -3 1 -4 0 4 -1 3 -2	1 1 0 1
1-1 1 0	-1 -4 2 3 0 -3 -2 4 1	4 -2 1 -3 0 3 -1 2 -4	0 1-1 1
1-1 1 1	1 -4 3 2 0 -2 -3 4 -1	-3 2 1 4 0 -4 -1 -2 3	-1-1 1-1
1 0-1-1	-4 -2 -3 1 0 -1 3 2 4	3 4 2 -1 0 1 -2 -4 -3	1 0-1-1
1 0-1 1	-3 -2 -4 -1 0 1 4 2 3	-2 -4 -3 -1 0 1 3 4 2	1 0 1 1
1 0 0-1	-2 -3 -4 1 0 -1 4 3 2	2 3 4 -1 0 1 -4 -3 -2	1 0 0-1
1 0 1-1	-3 -4 -2 1 0 -1 2 4 3	4 2 3 -1 0 1 -3 -2 -4	1 0 1-1
1 0 1 1	-2 -4 -3 -1 0 1 3 4 2	-3 -2 -4 -1 0 1 4 2 3	1 0-1 1
1 1-1 0	4 -2 1 -3 0 3 -1 2 -4	-1 -4 2 3 0 -3 -2 4 1	0-1 1 1
1 1-1 1	3 -2 -1 -4 0 4 1 2 -3	-1 4 -3 -2 0 2 3 -4 1	-1 1-1-1
1 1 0-1	4 -3 -1 -2 0 2 1 3 -4	-1 3 -2 -4 0 4 2 -3 1	1 1 0-1
1 1 0 1	2 -3 1 -4 0 4 -1 3 -2	-1 -3 4 2 0 -2 -4 3 1	1-1 0 1
1 1 1-1	3 -4 1 -2 0 2 -1 4 -3	-1 -2 3 4 0 -4 -3 2 1	-1-1-1 1
1 1 1 0	2 -4 -1 -3 0 3 1 4 -2	-1 2 -4 -3 0 3 4 -2 1	0 1 1-1
$f(y) \otimes_k Y_k$		$w \otimes p^{n-1-k}$	
$f(y)$		-4 -3 -2 -1 0 1 2 3 4	
$f(y) \otimes Y_1$		-2 -4 -3 1 -1 0 4 2 3	
$f(y) \otimes Y_0$		2 3 4 -4 -3 -2 -1 0 1	
$f(y) \otimes Y_1$		-3 -2 -4 0 1 -1 3 4 2	
$f(y) \otimes Y_0$		-1 0 1 2 3 4 -4 -3 -2	
$f(y) \otimes Y_0 \otimes Y_1$		4 2 3 -2 -4 -3 1 -1 0	
$f(y) \otimes Y_0 \otimes Y_1$		3 4 2 -3 -2 -4 0 1 -1	
$f(y) \otimes Y_0 \otimes Y_1$		1 -1 0 4 2 3 -2 -4 -3	
$f(y) \otimes Y_0 \otimes Y_1$		0 1 -1 3 4 2 -3 -2 -4	

John R. Miller

Communications Sciences Division, Code 7572
Naval Research Laboratory
Washington, D. C. 20375

Abstract -- It is proposed that a particular infinite-valued propositional calculus be used to define a retrieval function for a document retrieval system in which queries are well-formed formulas of the propositional calculus. With this approach, "weighted" queries may be processed in a simple, straightforward manner; queries may be represented as highly sparse vectors, and this representation may be transformed easily into the more conventional truth-table representation, and conversely; feedback techniques may be used; and a means is provided for circumventing the "independence assumption" with regard to subject identifiers.

Document Retrieval Systems, Local and Nonlocal Models

Consider a library consisting of a collection of documents. A person availing himself of the services of the library is usually interested in only a subset of the collection. That is, the library user wishes to read those and only those documents belonging to this subset. It is the job of a document retrieval system to determine as precisely as possible just what this subset is. Regardless of whether the document retrieval system consists of a group of human beings or a piece of electromechanical hardware or a computer program (or a combination of all of these), the fundamental operations comprising the retrieval process are the same: The library user, employing some suitable language, describes his interest. This description is called the query. Each document in the library is characterized by a document description in a data base accessible to the document retrieval system. The system searches the data base to determine which documents satisfy the query. A part of the system called the retrieval function (also known as the "matching" or "correlation" function) makes this determination.

The basic structure of a document retrieval system may be represented by an abstract model¹¹ in which mathematical objects play the roles of queries, the document descriptions, the retrieval function, and the documents themselves. Suppose

*This work was supported in part by Naval Electronics Systems Command Contract N00039-71-C-0255 and by National Science Foundation Grant GK-10656 while the author was a graduate student in the Electrical Engineering and Computer Sciences Department, University of California, Berkeley.

the library collection consists of m distinct documents. Let a one-to-one correspondence be established between the set of documents and the set of integers $\{1, 2, \dots, m\}$. The document associated with the integer j (where $1 \leq j \leq m$) shall be called the j th document. The j th document is characterized by the document description $D[j]$. (Following the Algol convention, square brackets denote subscripts.) Thus the data base is represented by $\{D[1], D[2], \dots, D[m]\}$. The user's query is denoted by Q , and the retrieval function is denoted by f .

The document retrieval model is either local or nonlocal depending on how the domain and range of f are defined. Let \underline{Q} denote the set of all permissible (gramatically correct) queries, \underline{Q} . (Throughout this paper, underlined symbols will be used to denote sets.) Let \underline{D} represent the set of all possible data bases. That is, members of \underline{D} are sets of the form $\{D[1], D[2], \dots, D[m]\}$, whose members are grammatically correct document descriptions. Finally, let \underline{R} denote the set of all subsets of $\{1, 2, \dots, m\}$. The nonlocal model is obtained if f is defined to be a mapping of the form $f: \underline{Q} \times \underline{D} \rightarrow \underline{R}$. The value of $f(Q, \{D[1], D[2], \dots, D[m]\})$ represents the set of retrieved documents appropriate to the given query Q and data base $\{D[1], D[2], \dots, D[m]\}$. The word "nonlocal" is used in this case because the decision of whether or not to retrieve a particular document is based upon a consideration of the descriptions of all documents. W. S. Cooper² presents a persuasive argument that in order to do document retrieval correctly, one should use the nonlocal model. Unfortunately, however, with our present state of knowledge, it appears difficult and costly to implement a document retrieval system which is an instance of the nonlocal model. Consequently, one is usually forced to use the local model, which is less general. The local model is obtained if f is defined to be a mapping of the form $f: \underline{Q} \times \{D[1], D[2], \dots, D[m]\} \rightarrow \{0, 1\}$. The value of $f(Q, D[j])$ is 1 if the j th document should be retrieved, 0 otherwise. The word "local" is used in this case because the decision of whether or not to retrieve a particular document is based upon a consideration of the description of that document alone. For a given query Q , the retrieval function f is in effect the characteristic function of the set of retrieved documents. C. C. Chang,¹ L. A. Zadeh,¹⁴ S. Watanabe,¹³ and others have considered generalizing the characteristic function of a set by replacing the two-element range $\{0, 1\}$

with the closed unit interval $I = \{x | x \text{ is a real number, and } 0 \leq x \leq 1\}$. The remainder of this paper will use the local model with this generalization. Under this interpretation, the value of $f(Q, D[j])$ may range continuously from 0 through 1 -- the larger the value, the stronger the indication that the j th document should be retrieved. The set of retrieved documents may be the set of all documents whose retrieval function values exceed some specified threshold, or it may be the set of μ documents having the μ highest retrieval function values, where μ is a specified integer less than or equal to m . Examples of document retrieval systems which operate in this fashion are SMART,² developed at Harvard and Cornell, and LABSEARCH,⁵ developed at the University of California, Berkeley.

An Instance of the Local Model Using a Particular Infinite-Valued Propositional Calculus

A popular choice for a query language is the language of well-formed formulas of the propositional calculus. The BNF grammar below describes such a language.

```

<query> ::= (<conjunction> | <query> v <conjunction>)
<conjunction> ::= (<negation> | <conjunction> ^ <negation>)
<negation> ::= ~ <negation> | <elemental sentence>
<elemental sentence> ::= (<query>)
                        | <subject identifier>
                        | <constant>

```

A <subject identifier> is a string representing a sentence of the form: "The document is about _____," where the blank (_____) contains a word or phrase naming a particular subject area. The connective v means "inclusive or." The connective \wedge means "and." The modifier \sim means "not." A <constant> is a string of digits and a decimal point representing a number in the closed unit interval I . In effect a constant stands for a sentence whose truth value is the number symbolized by the constant. The idea of including constants was suggested by the LABSEARCH system at Berkeley. In that document retrieval system, users may "multiply" subject identifiers and subexpressions by "weights" to reduce the effect of parts of the query considered to be of lesser importance.⁷ The use of a <constant>, usually with the \wedge connective, is a more general way of accomplishing the same thing.

Let Q be the set of all strings consistent with the above grammar, and consider a query $Q \in Q$. The query Q is really an abbreviation for the sentence: "I am interested in any document such that Q ." For example, suppose a library user is interested in documents concerning air pollution or coal but wishes to exclude those documents which are also about oil. He might word his query as follows: "I am interested in any document such that: ('The document is about air pollution.' v 'The document is about coal.') \wedge ~ 'The document is about oil.'" Instead of writing the entire sentence, "The document is about _____," one may abbreviate

it by writing only the word or phrase which fills the blank. The above query becomes: (air pollution v coal) \wedge ~ oil. Thus, for example, the phrase "air pollution" is a subject identifier representing the sentence, "The document is about air pollution." Let there be a finite vocabulary $\{S[1], S[2], \dots, S[n]\}$, from which subject identifiers may be chosen. That is,

$\langle \text{subject identifier} \rangle ::= S[1] | S[2] | \dots | S[n]$

With respect to the j th document, let $d[j, k]$ be the truth value of the sentence represented by $S[k]$. In other words, $d[j, k]$ is the truth value of the sentence, "The j th document is about $S[k]$." If an infinite-valued propositional calculus^{1,2} is being used, $d[j, k] \in I$. The value 1 denotes "true;" 0 denotes "false;" and intermediate values indicate truth values somewhere between absolute truth and absolute falsity.

Let $D[j]$, the document description characterizing the j th document be defined as follows: $D[j] = (d[j, 1], d[j, 2], \dots, d[j, n])$. $D[j]$ is an n -component vector, each component of which is in the closed unit interval I . This method of describing a document has been suggested by Maron and Kuhns as a consequence of probability theory⁶ and by Tahani as a consequence of fuzzy set theory.¹¹ The "document concept vector" used in the SMART document retrieval system is basically this kind of representation.¹⁰

A propositional calculus provides a means for assigning a truth value to a well-formed formula. The formula and the truth values of its elemental constituents are mapped into a truth value for the whole formula. This mapping may be effected by the retrieval function $f: Q \times \{D[1], D[2], \dots, D[m]\} \rightarrow I$. In this case the propositional calculus is infinite-valued because the truth values of the elemental constituents (the components of a given $D[j]$) and the truth value of the whole formula are permitted to be in the closed interval I rather than being restricted to the two-element set $\{0, 1\}$. There are many acceptable candidates for f .^{1,2} The one which will be presented below has a number of desirable properties for document retrieval.

If $Q \in Q$, the set of all well-formed formulas consistent with the BNF grammar presented above, and if $D[j] \in \{D[1], D[2], \dots, D[m]\}$, the set of document descriptions for the entire library collection, then let the retrieval function be defined as follows:

$$f(Q, D[j]) = g(Q) \cdot h(D[j])$$

where the dot (\cdot) denotes the vector dot (inner) product, and g and h are vector-valued functions defined below. In the definitions below, raising to a power is denoted by t , and multiplication of two scalars is denoted by $*$.

$$h(D[j]) = E[j] = (e[j, 0], e[j, 1], \dots, e[j, (2n)-1])$$

$$\text{where } e[j, i] = \prod_{k=1}^n (d[j, k]^t b[k-1, i])$$

and where $b[k,i]$ is the k th bit (counting from right to left, beginning with 0) in the binary representation of the integer i . That is (using the definition $x \div 0 = 1$ for any number x), $b[k,i] = (i \div (2^k)) \bmod 2$, where \div denotes division in which the fractional part of the quotient is discarded. Let $E[j]$ be called the "expanded document description" for the j th document. For example,

$$E[j] = (e[j,0], e[j,1], e[j,2], e[j,3], e[j,4], e[j,5], e[j,6], e[j,7], \dots)$$

$$= (1, d[j,1], d[j,2], d[j,1] * d[j,2], d[j,3], d[j,1] * d[j,3], d[j,2] * d[j,3], d[j,1] * d[j,2] * d[j,3], \dots)$$

Let P and Q be queries. That is, $P \in Q$ and $Q \in Q$. The function g is defined recursively:

$$g(P \wedge Q) = g(P) \diamond g(Q)$$

$$g(P \vee Q) = g(P) + g(Q) - g(P) \diamond g(Q)$$

$$g(\sim Q) = (w[0], w[1], \dots, w[(2^n)-1]) - g(Q)$$

$$\text{where } w[i] = \begin{cases} 1 & \text{if } i=0 \\ 0 & \text{if } i \neq 0 \end{cases}$$

$$g((Q)) = g(Q)$$

$$g(S[k]) = (w[0], w[1], \dots, w[(2^n)-1])$$

$$\text{where } w[i] = \begin{cases} 1 & \text{if } i = 2^k(k-1) \\ 0 & \text{if } i \neq 2^k(k-1) \end{cases}$$

$$g(\langle \text{constant} \rangle) = (w[0], w[1], \dots, w[(2^n)-1])$$

$$\text{where } w[i] = \begin{cases} c & \text{if } i=0 \\ 0 & \text{if } i \neq 0 \end{cases}$$

where c is the number represented by $\langle \text{constant} \rangle$. The operator \diamond denotes a new kind of vector product which shall be named the "diamond product." It is defined by the following Algol 60 procedure.

procedure Diamond (u, v, w, n);

real array u, v, w ; integer n ;

comment This procedure computes the diamond product of the real vectors v and w . The result is stored in the real vector u . The integer parameter n specifies the number of components in these vectors according to the array declaration real array $u, v, w[0:(2^n)-1]$. The only parameter whose contents are altered by the execution of this procedure is u . This procedure assumes the existence of an integer-valued function $\text{bitor}(i, j)$, which has as its value that integer whose binary representation is the bit-by-bit logical "or" of the binary representations of the integer arguments i and j . That is, each bit in the binary representation of the value of $\text{bitor}(i, j)$ is the logical "or" of the corresponding bits in the binary representations of i and j (for example, $\text{bitor}(1, 1)$ returns a value of 1, $\text{bitor}(0, 1)$ returns a value of 1, $\text{bitor}(1, 2)$ returns a value of 3, $\text{bitor}(5, 3)$ returns a value of 7);

```
begin
  integer i, j, max, lambda;
  max := (2^n)-1;
  for i := 0 step 1 until max do u[i] := 0.0;
  for i := 0 step 1 until max do
    begin
      for j := 0 step 1 until max do
        begin
          lambda := bitor(i, j);
          u[lambda] := v[i]*w[j] + u[lambda]
        end
      end
    end
  end Diamond
```

The effect of this procedure is as follows:

$$u[0] := v[0]*w[0];$$

$$u[1] := v[0]*w[1] + v[1]*w[0] + v[1]*w[1];$$

$$u[2] := v[0]*w[2] + v[2]*w[0] + v[2]*w[2];$$

$$u[3] := v[0]*w[3] + v[1]*w[2] + v[1]*w[3] + v[2]*w[1] + v[2]*w[3] + v[3]*w[0] + v[3]*w[1] + v[3]*w[2] + v[3]*w[3];$$

$$u[4] := v[0]*w[4] + v[4]*w[0] + v[4]*w[4];$$

$$u[5] := v[0]*w[5] + v[1]*w[4] + v[1]*w[5] + v[4]*w[1] + v[4]*w[5] + v[5]*w[0] + v[5]*w[1] + v[5]*w[4] + v[5]*w[5];$$

$$u[6] := v[0]*w[6] + v[2]*w[4] + v[2]*w[6] + v[4]*w[2] + v[4]*w[6] + v[6]*w[0] + v[6]*w[2] + v[6]*w[4] + v[6]*w[6];$$

$$u[7] := v[0]*w[7] + v[1]*w[6] + v[1]*w[7] + v[2]*w[5] + v[2]*w[7] + v[3]*w[4] + v[3]*w[7] + v[4]*w[3] + v[4]*w[7] + v[5]*w[2] + v[5]*w[7] + v[6]*w[1] + v[6]*w[7] + v[7]*w[0] + v[7]*w[1] + v[7]*w[2] + v[7]*w[3] + v[7]*w[4] + v[7]*w[5] + v[7]*w[6] + v[7]*w[7];$$

and so on for $u[8], u[9], u[10], \dots, u[(2^n)-1]$.

The retrieval function f which has been defined above is related to the "relevance" calculation procedure which Maron and Kuhns derived using probability theory,² to the infinite-valued propositional calculus "E II" described by Tsichritzis,^{1a} and to a scheme developed by Cziffra.³ However it is not identical to any of these.

An Example of the Evaluation of the Retrieval Function

Let $Q = (.7 \wedge S[1] \vee \sim S[2]) \wedge S[3]$, which means that the user is interested in any document which is about $S[3]$ and is either about $S[1]$ or not about $S[2]$, where the property of "being about $S[1]$ " is considered only .7 as important as "not being about $S[2]$." Without loss of generality, one may let $n=3$ because, for this particular Q , all components of $g(Q)$ with index higher than 7 will be 0. The function $g(Q)$ is evaluated as follows:

$$\begin{aligned}
&g((.7 \wedge S[1] \vee \sim S[2]) \wedge S[3]) \\
&=g((.7 \wedge S[1] \vee \sim S[2])) \diamond g(S[3]) \\
&=g(.7 \wedge S[1] \vee \sim S[2]) \diamond (0,0,0,0,1,0,0,0) \\
&=(g(.7 \wedge S[1]) + g(\sim S[2]) - g(.7 \wedge S[1])) \diamond g(\sim S[2]) \\
&\quad \diamond (0,0,0,0,1,0,0,0) \\
&=(g(.7) \diamond g(S[1]) + ((1,0,0,0,0,0,0,0) - g(S[2]))) \\
&\quad - g(.7) \diamond g(S[1]) \diamond ((1,0,0,0,0,0,0,0) - g(S[2]))) \\
&\quad \diamond (0,0,0,0,1,0,0,0) \\
&=((.7,0,0,0,0,0,0,0) \diamond (0,1,0,0,0,0,0,0) \\
&\quad + ((1,0,0,0,0,0,0,0) - (0,0,1,0,0,0,0,0)) \\
&\quad - (.7,0,0,0,0,0,0,0) \diamond (0,1,0,0,0,0,0,0) \\
&\quad \diamond ((1,0,0,0,0,0,0,0) - (0,0,1,0,0,0,0,0))) \\
&\quad \diamond (0,0,0,0,1,0,0,0) \\
&=((.7,0,0,0,0,0,0,0) \diamond (0,1,0,0,0,0,0,0) \\
&\quad + (1,0,-1,0,0,0,0,0) \\
&\quad - (.7,0,0,0,0,0,0,0) \diamond (0,1,0,0,0,0,0,0) \\
&\quad \diamond (1,0,-1,0,0,0,0,0)) \diamond (0,0,0,0,1,0,0,0) \\
&=((0,.7,0,0,0,0,0,0) + (1,0,-1,0,0,0,0,0) \\
&\quad - (0,.7,0,-.7,0,0,0,0)) \\
&\quad \diamond (0,0,0,0,1,0,0,0) \\
&=((1,0,-1,.7,0,0,0,0) \diamond (0,0,0,0,1,0,0,0) \\
&=(0,0,0,0,1,0,-1,.7)
\end{aligned}$$

Therefore,
 $f((.7 \wedge S[1] \vee \sim S[2]) \wedge S[3], D[j])$
 $=g((.7 \wedge S[1] \vee \sim S[2]) \wedge S[3]) \cdot h(D[j])$
 $= (0,0,0,0,1,0,-1,.7) \cdot (1, d[j,1], d[j,2],$
 $d[j,1] \cdot d[j,2], d[j,3], d[j,1] \cdot d[j,3], d[j,2] \cdot d[j,3],$
 $d[j,1] \cdot d[j,2] \cdot d[j,3])$
 $= d[j,3] - d[j,2] \cdot d[j,3] + .7 \cdot d[j,1] \cdot d[j,2] \cdot d[j,3]$

For example, if the document description of the j th document is $D[j] = (d[j,1], d[j,2], d[j,3]) = (.5, .3, .9)$, then the retrieval function value for the j th document with respect to the given query is

$$\begin{aligned}
&f((.7 \wedge S[1] \vee \sim S[2]) \wedge S[3], (.5, .3, .9)) \\
&=.9 - .3 \cdot .9 + .7 \cdot .5 \cdot .3 \cdot .9 \\
&=.7245
\end{aligned}$$

Some Properties of this Retrieval Scheme

One property of the retrieval function defined above is that it is separated into two functions, g and h , the former depending only on Q and the latter only on $D[j]$. Thus the query Q need be parsed only once, and the resulting vector $g(Q)$ may be saved and successively dot multiplied by the vectors $h(D[1]), h(D[2]), \dots, h(D[m])$ to obtain the retrieval function values for each of the m documents.

Another property is that identical queries are mapped into the same vector by g . For example, $g(Q \wedge Q) = g(Q)$. Thus, $f(Q \wedge Q, D[j]) = f(Q, D[j])$, provided Q contains no constants.

A third property is that $g(Q)$ is usually very sparse; that is, very few of the components of $g(Q)$ are nonzero. Therefore $g(Q)$ can be represented in a fashion which consumes very little storage in a computer's memory. For purposes of comparison, consider the truth table of Q , represented here by the vector-valued function $T(Q)$, defined as follows:

$$T(Q) = (t[0], t[1], \dots, t[(2^n) - 1])$$

where $t[i] = g(Q) \cdot h((b[0,i], b[1,i], \dots, b[n-1,i]))$

Thus, for example,

$$T((.7 \wedge S[1] \vee \sim S[2]) \wedge S[3]) = (0,0,0,0,1,1,0,.7)$$

if $n=3$. Both $g(Q)$ and $T(Q)$ are representations of the semantics of the query Q . If Q contains k distinct subject identifiers (and can not be transformed into an identical query containing fewer than k distinct subject identifiers) then $g(Q)$ will have at most 2^k nonzero components, while (if $Q \neq 0$) the truth table $T(Q)$ will have at least $2^{(n-k)}$ nonzero components. In a typical medium to large scale document retrieval system, n , the size of the subject identifier vocabulary, is on the order of 1000 to 10000, while k is typically about 5. Therefore, $2^{(n-k)} \gg 2^k$, so that $g(Q)$ is usually a much more compact way of representing the semantics of Q than is $T(Q)$. However, to be fair to $T(Q)$, one should note that each of the components of $T(Q)$ is restricted to the closed unit interval $\underline{1}$ whereas components of $g(Q)$ may range over much larger intervals. Let $g(Q) = (w[0], w[1], \dots, w[(2^n) - 1])$. Then $w[0] \in \underline{1}$. If $i \neq 0$, then the lower bound on $w[i]$ is $-2^{(numone(i)-1)}$ and the upper bound is $2^{(numone(i)-1)}$, where $numone(i)$ is a function whose value is the number of ones in the binary representation of i . That is,

$$numone(i) = \sum_{k=0}^{\infty} b[k,i]$$

Given the components of $g(Q)$, one may easily calculate those of $T(Q)$ and vice versa. Define the set $\underline{Y}(i)$ to be $\underline{Y}(i) = \{j | \text{bitor}(i,j) = i\}$. Then,

$$t[i] = \sum_{j \in \underline{Y}(i)} w[j]$$

$$w[i] = \sum_{j \in \underline{Y}(i)} (\text{relpar}(i,j) \cdot t[j])$$

where $\text{relpar}(i,j)$ is a function which computes the "relative parity" of i and j . The value of $\text{relpar}(i,j)$ is $+1$ if the binary representations of i and j have the same parities (both even or both odd) and is -1 otherwise. Specifically, $\text{relpar}(i,j) = (1 - 2^{((numone(i) + numone(j)) \bmod 2)})$. For example,

$$\begin{aligned}
t[0] &= w[0] \\
t[1] &= w[0] + w[1] \\
t[2] &= w[0] + w[2] \\
t[3] &= w[0] + w[1] + w[2] + w[3] \\
t[4] &= w[0] + w[4] \\
t[5] &= w[0] + w[1] + w[4] + w[5] \\
t[6] &= w[0] + w[2] + w[4] + w[6] \\
t[7] &= w[0] + w[1] + w[2] + w[3] + w[4] + w[5] + w[6] + w[7]
\end{aligned}$$

and so on. Cziffra³ derived similar relationships. In the other direction,

$$\begin{aligned}
w[0] &= t[0] \\
w[1] &= -t[0] + t[1] \\
w[2] &= -t[0] + t[2] \\
w[3] &= t[0] - t[1] - t[2] + t[3] \\
w[4] &= -t[0] + t[4] \\
w[5] &= t[0] - t[1] - t[4] + t[5]
\end{aligned}$$

$$w[6] = t[0] - t[2] - t[4] + t[6]$$

$$w[7] = -t[0] + t[1] + t[2] - t[3] + t[4] - t[5] - t[6] + t[7]$$

and so on. Note that $y(i)$ has $2^{(\text{numone}(i))}$ members.

Some Candidates for Further Investigation

Since $g(Q)$ is a vector, several of the techniques already developed in connection with extant document retrieval systems (for example, SMART) for processing vector queries may be applied to queries that are well-formed formulas of the propositional calculus. In particular, feedback from the user may be employed to effectively modify the query in an attempt to improve the performance of the retrieval system.¹⁰ Many of the concepts of the theory of adaptive pattern classification⁴ seem to be applicable.⁸

Under the probabilistic interpretation of the infinite-valued propositional calculus, each subject identifier $S[j]$ is regarded as an event, the probability of which is taken to be the truth value of the sentence which $S[j]$ represents. The theory requires that if $i \neq j$, then $S[i]$ and $S[j]$ must be independent, a condition which may not always be true in practice.⁵ One way to circumvent this difficulty might be to redefine the expanded document description $E[j]$ such that, for example, $e[j,3]$ would be the truth value of $S[1] \wedge S[2]$ with respect to the j th document; $e[j,7]$ would be the truth value of $S[1] \wedge S[2] \wedge S[3]$ with respect to the j th document, and so forth. The data base would consist of $\{E[1], E[2], \dots, E[m]\}$ instead of $\{D[1], D[2], \dots, D[m]\}$. A heavy price has to be paid in storage however, since $E[j]$ has 2^n components while $D[j]$ has only n . One way to reduce the penalty would be to restrict the number of distinct subject identifiers that may appear in a query to some small number k , for then any $e[j,i]$ such that $\text{numone}(i) > k$ will never be used and may be discarded. Furthermore, for any one particular document, the majority of subject identifiers will have no applicability, so that any $e[j,i]$ which depends on one or more of them may be eliminated. In general, if λ is the number of subject identifiers which have nonzero applicability to the j th document, and if k is the maximum number of distinct subject identifiers permitted in a query, then the number of components of $E[j]$ which need to be saved is

$$\min(\lambda, k) \sum_{i=0} C(\lambda, i)$$

where $C(\lambda, i)$ is the number of combinations of λ things taken i at a time $(\lambda! / (i! * (\lambda - i)!))$. For example, if $\lambda = 10$ and $k = 4$, then storage must be provided for $C(10,0) + C(10,1) + C(10,2) + C(10,3) + C(10,4) = 176$ components. This is not a small number, but at least it is preferable to $2^{10} = 1024$, the number of components which would be required if no restrictions were placed on the number of subject identifiers in the query.

Acknowledgments

The author is very grateful to the following people, who provided advice, arranged for financial support, or arranged for the author's leave from work: William S. Cooper, Peter Cziffra, Lance J. Hoffman, Keh-Lon Lee, M. E. Maron, R. D. Misner, Frank A. Polkinghorn, Jr., Michael R. Stonebraker, Irene Travis, Bruce Wald, Eugene Wong, and Lotfi A. Zadeh. The author takes full responsibility for any errors or omissions in this paper.

References

- ¹ CHANG, C. C. Infinite Valued Logic as a Basis for Set Theory. Proceedings of the 1964 International Congress on the Logic, Methodology and Philosophy of Science. North-Holland Publishing Company. 1965. pages 93-100.
- ² COOPER, W. S. A Definition of Relevance for Information Retrieval. Information Storage and Retrieval. vol. 7, no. 1. June 1971. pages 19-37.
- ³ CZIFFRA, PETER. Weighted Boolean Requests. Unpublished report. May 1971.
- ⁴ IDE, ELEANOR. Relevance Feedback in an Automatic Document Retrieval System. Master's Thesis. Report ISR-15 to the National Science Foundation. Department of Computer Science, Cornell University, Ithaca, New York. January, 1969.
- ⁵ MARON, M. E., A. J. HUMPHREY, and J. C. MEREDITH. An Information Processing Laboratory for Education and Research in Library Science: Phase I. U. S. Department of Health, Education and Welfare, Office of Education, Bureau of Research. Interim Report, Project No. 7-1085. July, 1969.
- ⁶ MARON, M. E. and J. L. KUHN. On Relevance, Probabilistic Indexing and Information Retrieval. Journal of the Association for Computing Machinery. vol. 7, no. 3. July 1960. pages 216-244.
- ⁷ MIGNON, EDMOND and IRENE TRAVIS. IABSEARCH: ILR Associative Search System Terminal User's Manual. U. S. Department of Health, Education, and Welfare, Office of Education, Bureau of Research. Final Report, Project No. 7-1085. September, 1971.
- ⁸ MILLER, JOHN R. Pattern Classifiers as Simple Retrieval Systems. Ph. D. Dissertation. College of Engineering, University of California, Berkeley, California. December, 1974.
- ⁹ SALTON, GERARD. Automatic Information Organization and Retrieval. McGraw-Hill. 1968.
- ¹⁰ SALTON, GERARD (ed.). The SMART Retrieval System: Experiments in Automatic Document Processing. Prentice-Hall. 1971.

- ¹¹ TAHANI, VALIOLLAH. A General Conceptual Framework for Information Retrieval Systems. Ph. D. Dissertation. College of Engineering, University of California, Berkeley, California. September, 1972.
- ¹² TSICHRITZIS, D. Approximate Logic. Conference Record of the 1973 International Symposium on Multiple-Valued Logic. pages 205-216.
- ¹³ WATANABE, SATOSI. Modified Concepts of Logic, Probability, and Information Based on Generalized Continuous Characteristic Function. Information and Control. vol. 15, no. 1. July, 1969. pages 1-21.
- ¹⁴ ZADEH, L. A. Fuzzy Sets. Information and Control. vol. 8, no. 3. June, 1965. pages 338-353.

MATHEMATICAL PROPERTIES OF BOOLEAN TRANSFORMATIONS*

Ytzhak Leventel

Bell Telephone Laboratories
Naperville, Illinois 60540

Melvin A. Breuer

Departments of Electrical Engineering
and Computer Science
University of Southern California
Los Angeles, California 90007

ABSTRACT

This paper is concerned with the study of special aspects of the transformation $\underline{Y} = f(\underline{y}, \underline{x})$ which occurs as the "next state function" in the study of sequential circuits. To study this function we employ three models, namely a Boolean equation form, a Boolean matrix form, and a linear matrix form. The latter is based upon the Reed-Muller expansion of a Boolean function. Our major results deal with the study of the interrelationships between these models as well as the computation of path and cycle lengths associated with state transitions.

I. INTRODUCTION

A sequential circuit has the general form shown in Figure 1, where $\underline{x} = (x_1, x_2, \dots, x_n)$ is the input vector, $\underline{z} = (z_1, z_2, \dots, z_m)$ the output vector, and $\underline{y} = (y_1, y_2, \dots, y_n)$ and $\underline{Y} = (Y_1, Y_2, \dots, Y_n)$ are the current and next state vectors respectively.

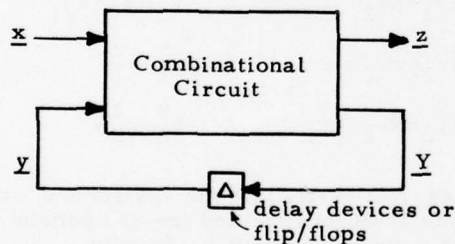


Figure 1. General structure of a sequential circuit.

The state operation of the circuit is defined by the transformation

$$\underline{Y} = f(\underline{y}, \underline{x}).$$

This paper deals with various mathematical models for studying the state transition mapping (transformation) process which occurs in finite state machines. The major difference between the operation of this transformation for the case of synchronous and asynchronous circuits is that for the former, the application of the transform occurs once per clock pulse, while for the latter the transformation may occur several times, the

* This work was supported by the Office of Naval Research, under Grant No. N00014-67-A-0069-0019 (NR048-299).

exact number being determined by the functional correspondence between \underline{y} and \underline{Y} itself. That is, \underline{Y} may change several times before the circuit stabilizes, denoted by the condition $\underline{y} = \underline{Y}$.

To study the characteristics of the transformation from \underline{y} to \underline{Y} we will present and analyze three forms for this mapping, namely

1. a Boolean equation form
2. a Boolean matrix form
- and
3. a linear matrix form.

Many applications of these three forms to the generation of fault detection tests and synchronizing sequences are given in reference 7.

II. MATHEMATICAL BACKGROUND

In this section we present some of the basic definitions and results which will be used in the subsequent sections. For a more detailed discussion of this material the reader is referred to references 1, 2, and 3.

A Galois field is a field with a finite number q of elements. It is denoted by $GF(q)$. The Galois field $GF(2)$ will be of special interest in the subsequent sections. For this field, addition and multiplication corresponds to the exclusive-or and AND operations, respectively. The exclusive-or operation is denoted by \oplus . The number of elements of a Galois field is p^r , where p is prime and r is an integer. $GF(p)$ is said to be the ground field of $GF(p^r)$ and $GF(p^r)$ is an extension field of $GF(p)$. The order e of an element x of $GF(q)$ is the smallest integer for which

$$x^e = 1.$$

Given a square matrix N , the elements of which belong to F , the characteristic polynomial of the matrix is the determinant of $sI - N$ and is denoted by $\delta(s)$. For any square matrix N with elements in a field F , $\delta(N) = 0$, where 0 denotes the zero matrix (Cayley-Hamilton Theorem). The eigenvalues of N are the roots of $\delta(s) = 0$. If the elements of N are in $GF(p)$, then the eigenvalue may be in any extension field $GF(p^r)$. The eigenvector t_λ corresponding to the eigenvalue λ of N is a solution to the equation $Nt_\lambda = \lambda t_\lambda$. The minimal polynomial $u(s)$ of a square matrix is the polynomial of a minimal degree such that $u(N) = 0$.

III. BOOLEAN TRANSFORMS

The Boolean vector space of dimension n is defined as the set of n -tuples in β^n , where β is the set $\{0, 1\}$ of elements of a switching algebra.⁴ It is denoted by $(\beta^n; \beta)$.

Given a Boolean vector space, a Boolean transform between two vectors \underline{y} and \underline{Y} of the space $(\beta^n; \beta)$ is defined by the equation

$$\underline{Y} = f(\underline{y}, \underline{x}) \quad (1)$$

where f is a vector of n Boolean (switching) functions and \underline{x} is a parameter vector of dimension m .

The transform can be denoted by

$$\underline{Y} \triangleq \mathcal{F}(\underline{y}) \quad (2)$$

or equivalently

$$\mathcal{F}: \underline{y} \xrightarrow{f(\cdot, \underline{x})} \underline{Y}. \quad (3)$$

The assignments of \underline{x} are taken from $(\beta^m; \beta)$. The ℓ -th power of a transform \mathcal{F} is a transform \mathcal{F}^ℓ which is equivalent to ℓ successive applications of \mathcal{F} on the vector \underline{y} . It is denoted by the equation

$$\underline{Y}^\ell = \mathcal{F}^\ell(\underline{y}) \quad (4)$$

and is defined by the equation

$$\mathcal{F}^\ell(\underline{y}) = \mathcal{F}(\mathcal{F}(\mathcal{F}(\dots(\underline{y})\dots))). \quad (5)$$

Theorem 1: Given a vector $\underline{y} \in (\beta^n; \beta)$, and a transform \mathcal{F} , then there exist two integers p and c , such that

$$\mathcal{F}^{p+c}(\underline{y}) = \mathcal{F}^p(\underline{y}) \quad (6)$$

where p and c are called the path and the cycle lengths respectively.

Proof: Suppose we apply \mathcal{F} to \underline{y} an unlimited number of times. Since there are 2^n possible vectors in $(\beta^n; \beta)$ some power of \mathcal{F} must be equal to a previously generated vector. ■

Corollary 1: The least upper bound for p is 2^n . ■

Theorem 2: The least upper bound for c , denoted by C_n , is given by

$$C_n = \prod (p_i)^j \quad (7)$$

where p_i and j are selected such that p_i is a prime number and j is the maximum power such that

$$(p_i)^j \leq 2^n. \quad (8)$$

The product in Eq. (7) is taken over all p_i which satisfy inequality (8).

Proof: The cycle length c is given by the least common multiple of all the cycle lengths corresponding to all of the possible assignments of \underline{x} . The contribution of the individual cycles will be maximized by taking the maximum powers of prime numbers. Since the largest cycle length is 2^n , this explains the restriction

$$(p_i)^j \leq 2^n.$$

C_n is the least upper bound since the proof is constructive. ■

Example 1: Let us consider the transform*

$$Y_1 = y_2(x + \bar{y}_1)$$

$$Y_2 = \bar{y}_2 + x\bar{y}_1.$$

Below we indicate the different powers of the transform:

$y_1^1 = y_2(x + \bar{y}_1)$	$y_1^5 = x\bar{y}_2 + \bar{x}y_2 + x\bar{y}_1$
$y_2^1 = \bar{y}_2 + x\bar{y}_1$	$y_2^5 = \bar{y}_2 + xy_1$
$y_1^2 = \bar{y}_2 + x\bar{y}_1$	$y_1^6 = \bar{y}_2 + xy_1$
$y_2^2 = \bar{x}y_2 + y_1y_2 + x\bar{y}_2$	$y_2^6 = y_2$
$y_1^3 = \bar{x}y_2 + y_1y_2 + x\bar{y}_2$	$y_1^7 = y_2$
$y_2^3 = \bar{x}\bar{y}_2 + xy_2$	$y_2^7 = \bar{y}_2 + x\bar{y}_1$
$y_1^4 = \bar{x}\bar{y}_2 + xy_2$	$y_1^8 = y_1^2$
$y_2^4 = x\bar{y}_2 + \bar{x}y_2 + x\bar{y}_1$	$y_2^8 = y_2^2.$

It is easy to verify that the assignment $\underline{x} = 0$ produces a cycle of length 2 and the assignment $\underline{x} = 1$ produces a cycle of length 3. Finally,

$$p = 2 \quad \text{and} \quad c = 6. \quad \blacksquare$$

Theorem 3:⁹ For large n we have

$$C_n > e^{2^{n-1}}. \quad \blacksquare$$

IV. MINTERM VECTOR FORM (MVF)

A product term of the form $y_1^{e_1}y_2^{e_2}\dots y_n^{e_n}$ is said to be a minterm over the variables in $\underline{y} = (y_1, y_2, \dots, y_n)$, where

$$(e_1, \dots, e_n) \in (\beta^n; \beta) \quad \text{and} \quad y^e = \begin{cases} y & \text{if } e = 1 \\ \bar{y} & \text{if } e = 0 \end{cases}.$$

* We will represent Boolean transforms by switching expressions.

For example $y_1 \bar{y}_2 y_3$ is a minterm and corresponds to $(1, 0, 1) \in (\beta^3: \beta)$. Since

$$\sum_{i=1}^n e_i 2^{n-i} = 5,$$

we say that this minterm corresponds to the decimal 5. The minterm vector form (MVF) of a vector y , denoted by \underline{m} , is a column vector consisting of all of the minterms over y . The components in \underline{m} are ordered by their corresponding decimal numbers.

Example 2: For a vector y of dimension 3, we have

$$y = (y_1, y_2, y_3)$$

and

$$\underline{m} = (\bar{y}_1 \bar{y}_2 \bar{y}_3, \bar{y}_1 \bar{y}_2 y_3, \bar{y}_1 y_2 \bar{y}_3, \bar{y}_1 y_2 y_3, y_1 \bar{y}_2 \bar{y}_3, y_1 \bar{y}_2 y_3, y_1 y_2 \bar{y}_3, y_1 y_2 y_3).$$

The transform between y and \underline{m} is called the decoding of the vector y and is denoted by

$$\underline{m} = \mathcal{D}(y). \quad (9)$$

This is a transform of an n -dimensional vector into a vector of dimension 2^n .

Equation 1 can be written as

$$Y_j = f_j(y, x) \text{ for } j = 1, 2, \dots, n \quad (10)$$

or

$$Y_j = \underline{a}_j \cdot \underline{m} \quad (11)$$

where \underline{a}_j is a row vector of Boolean expressions over x , and the dot represents a scalar product.

The actual assignments of \underline{a}_j belong to $(\beta^{2^n}: \beta)$.

The zero (one) vector, denoted by $\underline{0}$ ($\underline{1}$), of $(\beta^{2^n}: \beta)$ is a vector of size 2^n , the components of which are all zeros (ones).

Theorem 4: Given

$$Y_j = \underline{a}_j \cdot \underline{m}$$

and

$$Y_k = \underline{a}_k \cdot \underline{m}$$

then we have

$$\bar{Y}_j = \bar{\underline{a}}_j \cdot \underline{m} \quad (12)$$

$$Y_j Y_k = (\underline{a}_j \wedge \underline{a}_k) \cdot \underline{m} \quad (13)$$

and

$$Y_j + Y_k = (\underline{a}_j \vee \underline{a}_k) \cdot \underline{m} \quad (14)$$

where $\bar{\underline{a}}_j$ is the complement of \underline{a}_j , and " \wedge " and " \vee " are the AND and OR operations, respectively. These operations are carried out component by component.

The vectors of $(\beta^{2^n}: \beta)$ satisfy the postulates of a Boolean algebra and can be considered as the

representation of the function Y_j with respect to the base \underline{m} .

The matrix form Φ of the transform \mathcal{D} is defined by

$$\underline{Y} = \Phi \underline{m} \quad (15)$$

where Φ is of size $n \times 2^n$, the rows of which are defined by Eq. (11).

Example 3: Consider the transform of Example 1, namely

$$Y_1 = y_2(x + \bar{y}_1)$$

$$Y_2 = \bar{y}_2 + x \bar{y}_1.$$

We have

$$Y_1 = 0 \cdot \bar{y}_1 \bar{y}_2 + 1 \cdot \bar{y}_1 y_2 + 0 \cdot y_1 \bar{y}_2 + x \cdot y_1 y_2$$

$$Y_2 = 1 \cdot \bar{y}_1 \bar{y}_2 + x \cdot \bar{y}_1 y_2 + 1 \cdot y_1 \bar{y}_2 + 0 \cdot y_1 y_2$$

or equivalently

$$Y_1 = (0, 1, 0, x) \cdot \underline{m}$$

$$Y_2 = (1, x, 1, 0) \cdot \underline{m}.$$

Finally,

$$\underline{Y} = \begin{bmatrix} 0 & 1 & 0 & x \\ 1 & x & 1 & 0 \end{bmatrix} \cdot \underline{m}.$$

Equation (15) can be written in a symbolic form

$$\underline{Y} = \Phi \underline{m}.$$

Given

$$\underline{Y} = f(y, x)$$

$$\underline{m} = \mathcal{D}(y)$$

and

$$\underline{M} = \mathcal{D}(\underline{Y})$$

then \mathcal{Q} is the transform defined as

$$\underline{M} = \mathcal{Q}(\underline{m}). \quad (16)$$

Theorem 5: \underline{M} and \underline{m} are related by the equation

$$\underline{M} = A \underline{m} \quad (17)$$

where A is a $2^n \times 2^n$ matrix, the elements of which are Boolean switching expressions over the components of \underline{x} .

Proof: We have

$$\underline{Y} = \Phi \underline{m} = [\Phi_0, \Phi_1, \Phi_2, \dots, \Phi_t] \underline{m}$$

where $t = 2^n - 1$ and Φ_i are the columns of Φ . Therefore

$$\underline{M} = B(\underline{Y}) = [B(\underline{\Phi}_0), B(\underline{\Phi}_1), \dots, B(\underline{\Phi}_t)] \underline{m}. \quad \blacksquare$$

Example 4: Let us consider the result of Example 3, namely

$$\underline{Y} = \begin{bmatrix} 0 & 1 & 0 & x \\ 1 & x & 1 & 0 \end{bmatrix} \underline{m}.$$

We have

$$\underline{\Phi}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \quad B(\underline{\Phi}_0) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \bar{\alpha}_1 & \bar{\alpha}_2 \\ \alpha_1 & \alpha_2 \end{bmatrix}$$

$$\underline{\Phi}_1 = \begin{bmatrix} 1 \\ x \end{bmatrix} \quad B(\underline{\Phi}_1) = \begin{bmatrix} 0 \\ 0 \\ \bar{x} \\ x \end{bmatrix}$$

$$\underline{\Phi}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad B(\underline{\Phi}_2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\underline{\Phi}_3 = \begin{bmatrix} x \\ 0 \end{bmatrix} \quad B(\underline{\Phi}_3) = \begin{bmatrix} \bar{x} \\ 0 \\ x \\ 0 \end{bmatrix}.$$

Therefore

$$A = \begin{bmatrix} 0 & 0 & 0 & \bar{x} \\ 1 & 0 & 1 & 0 \\ 0 & \bar{x} & 0 & x \\ 0 & x & 0 & 0 \end{bmatrix}.$$

Theorem 6: The sum of each column of A is one. The product of two elements in the same column is zero.

Proof: The result follows from the fact that each row $B(\underline{\Phi}_i)$ corresponds to the minterms over $\underline{\Phi}_i$ and the fact that the sum of all minterms is one and the product of two minterms is zero. \blacksquare

The class of matrices defined by Eq. (17) and having the properties described by Theorem 6 are called **deterministic state transition matrices** (DSTM). The theory developed here for DSTM's applies to the classical concepts of state transition matrices and state diagrams used in finite state machine theory. The vector \underline{m} is a vector of minterms and only one component may be one at a time.

A specific vector \underline{m} for which the j -th component is one is said to define state j . From A we can construct a state transition diagram. An edge

connects state j to state i if the entry a_{ij} in A is non zero. This edge is labeled by a_{ij} .

The **logical multiplication** of two matrices is defined similar to that of algebraic matrix multiplication, except that multiplication and addition of scalars are replaced by AND and OR operations respectively.

Theorem 7: Let A' and A'' be two DSTM's, and $A''' = A' \times A''$. Then A''' is a DSTM. \blacksquare

Theorem 8: The transforms \mathfrak{F} and \mathcal{Q} are isomorphic.

Proof: This result follows since there is a one to one correspondence between \underline{y} and \underline{m} and \underline{Y} and \underline{M} . \blacksquare

The ℓ -th power of the transform \mathcal{Q} is the transform which is equivalent to ℓ successive applications of \mathcal{Q} . It is denoted by \mathcal{Q}^ℓ .

We have that

$$\underline{m}^\ell = \mathcal{Q}^\ell(\underline{m}). \quad (18)$$

$$\underline{m}^\ell = \mathcal{Q}^\ell \cdot \underline{m}. \quad (19)$$

Proof: By induction on ℓ . \blacksquare

Theorem 10: Given a vector $\underline{m} = B(\underline{y})$, a transform \mathcal{Q} and its matrix A , there exist two integers p and c such that

$$A^{p+c} = A^p.$$

Proof: The proof follows from Theorems 1 and 8. \blacksquare

V. REED MULLER FORM

In this section we shall study the implications of using the Reed Muller form⁶ on the transforms \mathfrak{F} and \mathcal{Q} which were studied in the previous sections.

The **Reed Muller state vector** (RMSV) corresponding to a vector $\underline{y} \in (\beta^n; \beta)$ is a column vector \underline{r} of size 2^n , where

$$\underline{r} = (1, y_1, y_2, \dots, y_n, y_1 y_2, y_1 y_3, \dots, y_{n-1} y_n, \dots, y_1 y_2 \dots y_{n-1}, \dots, y_2 y_3 \dots y_n, y_1 y_2 \dots y_n).$$

Theorem 11: \underline{m} and \underline{r} corresponding to the same vector \underline{y} are related by a linear change of variables over $GF(2)$, given by the matrix equation

$$\underline{m} = C \underline{r}.$$

Proof: By using the Reed Muller form, any element m_j can be expressed as

$$m_j = c_j \cdot \underline{r} \quad \text{where} \quad c_{ij} \in \{0, 1\}$$

and where the dot operator is the scalar product over GF(2).

Example 5: For $n = 2$ we have

$$\underline{m} = \begin{bmatrix} \bar{y}_1 \bar{y}_2 \\ \bar{y}_1 y_2 \\ y_1 \bar{y}_2 \\ y_1 y_2 \end{bmatrix} = \begin{bmatrix} (1 \oplus y_1)(1 \oplus y_2) \\ (1 \oplus y_1)y_2 \\ y_1(1 \oplus y_2) \\ y_1 y_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ y_1 \\ y_2 \\ y_1 y_2 \end{bmatrix} = C \underline{r}.$$

We also have

$$\underline{r} = C^{-1} \underline{m} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \underline{m}.$$

Given the transforms \mathcal{F} and \mathcal{Q} , it is possible to define \underline{r} and \underline{R} as follows:

$$\underline{r} = C^{-1} \underline{m} \quad \text{and} \quad \underline{R} = C^{-1} \underline{M}.$$

Theorem 12: Let \mathcal{B} be the transform relating \underline{r} and \underline{R} ; i.e.,

$$\underline{R} = \mathcal{B}(\underline{r}).$$

Then \mathcal{Q} and \mathcal{B} are isomorphic.

Proof: Similar to Theorem 8.

Theorem 13: There exist a matrix B , the elements of which belong to GF(2), such that

$$\underline{R} = B \underline{r}$$

and

$$B = C^{-1} A C \quad \text{or} \quad A = C B C^{-1}$$

where A is the matrix of the transform \mathcal{Q} of Theorem 12.

Proof: We have

$$\underline{M} = A \underline{m}$$

$$C \underline{R} = A C \underline{r}$$

and hence

$$\underline{R} = C^{-1} A C \underline{r} = B \underline{r}.$$

Theorem 14: Given \underline{r} and the matrix B of the transform \mathcal{B} as defined in Theorem 12, then there exist two integers p and c such that

$$B^{p+c} = B^p.$$

Proof: We have that

$$B^k = C^{-1} A^k C. \quad (20)$$

From Theorem 10 and Eq. (20) we get

$$B^{p+c} = B^p$$

where p and c are the same integers as in Theorem 10.

Example 6: Using the results of Example 4, we obtain

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \oplus x \\ 1 & 0 & 1 & 0 \\ 0 & 1 \oplus x & 0 & x \\ 0 & x & 1 & 0 \end{bmatrix}.$$

In Example 5 we obtained

$$C = C^{-1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Computing B we obtain

$$B = C^{-1} A C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \oplus x \\ 1 & 0 & 1 \oplus x & x \\ 0 & 0 & x & x \end{bmatrix}.$$

It can be shown that

$$B^8 = B^2.$$

VI. LINEAR ANALYSIS OF THE TRANSFORMS \mathcal{Q} AND \mathcal{B}

Jordan decomposition of linear matrices is a well-known result of linear algebra. Given a linear matrix N , it can be written as

$$N = T^{-1} J T$$

where J is a Jordan form of N . The Jordan matrix has the structure shown below

$$J = \begin{bmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ 0 & & & J_k \end{bmatrix}$$

where the J_i 's are square Jordan submatrices. For each eigenvalue λ of N there is at least one Jordan submatrix J_i of the form

$$J_i = \begin{bmatrix} \lambda & 1 & & 0 \\ 0 & \lambda & 1 & \\ 0 & & \ddots & 1 \\ \vdots & & & \ddots \\ 0 & & & & \lambda \end{bmatrix}.$$

In the following discussion we will denote a Jordan submatrix by J when no ambiguity occurs.

Theorem 15: Given a Jordan submatrix of size $j \times j$ corresponding to a zero eigenvalue, then

$$J^j = 0$$

and j is the smallest integer with this property. ■

Theorem 16: Given a Jordan submatrix J of size $j \times j$ corresponding to a nonzero eigenvalue λ , where λ belongs to the Galois field $GF(2^r)$ and λ is of order e , then

$$J^m = I$$

where $m = e2^\ell$ and ℓ is the smallest integer such that

$$j \leq 2^\ell.$$

Proof: It is well known (Galois field theory) that

$$(s+1)^{2^q} = s^{2^q} + 1 \text{ for all } q. \quad (21)$$

On the other hand e is the smallest integer such that

$$(s^e + 1) = (s + \lambda)p_\lambda(s)$$

where $p_\lambda(s)$ is a polynomial of s of degree $e-1$.

We have

$$(s^e + 1)^j = (s + \lambda)^j p_\lambda^j(s). \quad (22)$$

Obviously $(s^e + 1)^j$ divides $(s^e + 1)^{2^\ell}$ where ℓ is the smallest integer such that

$$j \leq 2^\ell.$$

Using Eq. (21)

$$(s^e + 1)^{2^\ell} = s^{e2^\ell} + 1.$$

Therefore, $(s^e + 1)^j$ divides $s^{e2^\ell} + 1$ and ℓ is the smallest integer for which this occurs.

Finally it is possible to write

$$(s^{e2^\ell} + 1) = (s^e + 1)^j p_j(s).$$

From Eq. (22) we obtain

$$(s^{e2^\ell} + 1) = (s + \lambda)^j p_\lambda^j(s) p_j(s). \quad (23)$$

If we substitute J for s in Eq. (23) we obtain

$$(J^{e2^\ell} + I) = (J + \lambda I)^j p_\lambda^j(J) p_j(J).$$

Since $(s + \lambda)^j$ is the minimal polynomial of J , it follows that

$$(J + \lambda I)^j = 0.$$

Therefore

$$J^{e2^\ell} = I. \quad \blacksquare$$

Theorem 17: Given a Jordan submatrix J with nonzero eigenvalue λ of order e , the matrices

$$0, G, G^2, G^3, \dots, G^{e-1}, G^e$$

form a finite field, where

$$G = J^{2^\ell}$$

and ℓ is defined as in Theorem 16.

Proof: We know that J satisfies its minimal polynomial, namely

$$(J + \lambda I)^j = 0.$$

Now $(s + \lambda)^j$ divides $(s + \lambda)^{2^\ell}$ and $(s + \lambda)^{2^\ell} = s^{2^\ell} + \lambda^{2^\ell}$ where ℓ is the smallest integer such that

$$s^{2^\ell} + \lambda^{2^\ell} = (s + \lambda)^j p_j(s). \quad (24)$$

It is known from Galois field theory that λ^{2^ℓ} is one of the ℓ -conjugates of λ and has the same order as λ .

By substituting J for s in Eq. (24) we have

$$G = J^{2^\ell} = \lambda^{2^\ell} I = gI.$$

The elements $0, g, g^2, \dots, g^{e-1}, 1$ form a field of $e+1$ elements. So will the different powers of G and the zero matrix.

Note that Theorem 16 is actually a corollary to Theorem 17 since

$$G^e = g^e I = I. \quad \blacksquare$$

Theorem 18: Given a Jordan matrix

$$J = \begin{bmatrix} J_1 & & & \\ & J_2 & & 0 \\ & & \ddots & \\ 0 & & & J_k \end{bmatrix}$$

the elements of which belong to $GF(q^r)$ then there exist two integers p and c such that

$$J^{p+c} = J^p$$

and p, c are defined as follows:

1. p is the size of the largest submatrix which has zero eigenvalues.
2. c is the least common multiple of m_1, m_2, \dots, m_k where

$$J_i^{m_i} = I.$$

Proof: The result follows from the fact that

$$J^L = \begin{bmatrix} J_1^L & & & \\ & J_2^L & & \\ & & \ddots & \\ & & & J_k^L \end{bmatrix}$$

and from Theorems 15 and 16. ■

Given a matrix A or B as defined in Sections IV and V, then the integers p and c of Theorems 10 and 14 can be obtained as described next.

Procedure 1: Calculation of p and c using matrices A or B .

Step 1. For each binary assignment x_i of x , perform the Jordan decomposition of the corresponding matrix A or B .

Step 2. Compute j_i and m_i such that

$$J_i^{j_i} = 0 \quad \text{and} \quad J_i^{m_i} = I$$

according to Theorems 15 and 16.

Step 3. p is the largest j_i obtained in Step 2. c is the least common multiple of all of the m_i obtained in Step 2. ■

The validity of this procedure follows from Theorem 18.

Example 7: Let us consider the matrix

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \oplus x \\ 1 & 0 & 1 \oplus x & x \\ 0 & 0 & x & x \end{bmatrix}.$$

For the assignment $x = 0$, we obtain

$$B(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The characteristic polynomial of $B(0)$ is $s^2(s+1)^2$ and the eigenvalues are 0 and 1.

We have

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad T^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$J = \begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix} = T^{-1}BT = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Therefore

$$J_1 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad J_1^2 = I$$

$$J_2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad J_2^2 = 0.$$

For the assignment $x = 1$ we obtain

$$B(1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

The characteristic polynomial of $B(1)$ is $s(s+1)(s^2+s+1)$ and its eigenvalues are 0, 1, a and b . Now

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & a & b \\ 0 & 1 & b & a \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ b & 0 & a & b \\ a & 0 & b & a \end{bmatrix}.$$

Therefore

$$J = \begin{bmatrix} J_3 & & & \\ & J_4 & & \\ & & J_5 & \\ & & & J_6 \end{bmatrix} = T^{-1}BT = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & b \end{bmatrix}$$

where

$$J_3 = 0 \quad J_4 = 1 \quad J_5 = a \quad J_6 = b$$

and

$$J_3^1 = 0 \quad J_4^1 = 0 \quad J_5^3 = 1 \quad J_6^3 = 1.$$

The result from this computation is

$$p = 2 \quad c = 6$$

as we found previously. ■

It is important to notice that Procedure 1 applies to the matrix B of Section V as well as to the

matrix A of Section IV, which are both derived from the transform \mathfrak{F} of Section III.

VII. CONCLUSION

We have defined a Boolean transform \mathfrak{F} by

$$\underline{Y} = f(\underline{y}, \underline{x})$$

and derived the isomorphic transforms \mathcal{A} and \mathcal{B} defined by the matrices A and B given in Sections IV and V. \mathfrak{F} is a relation between two binary n-dimensional vectors, \mathcal{A} is the relation between the two MVF's corresponding to the two previous vectors, and \mathcal{B} is the relation between the two corresponding RMSV's.

The complete relationship between \mathcal{A} , \mathcal{B} , and \mathfrak{F} is given in Figure 2.

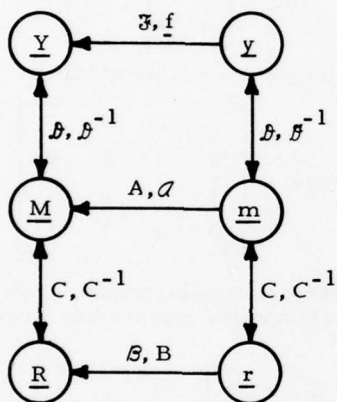


Figure 2. Relationship between \mathcal{A} , \mathcal{B} and \mathfrak{F} .

The transforms \mathcal{B} and \mathcal{C} are reversible, i.e., given any one of the three vectors, \underline{y} , \underline{m} , or \underline{r} , we can compute the other two.

The transforms \mathfrak{F} , \mathcal{A} , and \mathcal{B} are not necessarily reversible. Whenever one transform is reversible, so are the other two, and both matrices A and B will be non-singular. Singularity of A and B means irreversibility of the transforms.

There are two basic schools of research which employ multi-valued logic to digital circuits. One school deals with the realization and related design algorithms for multi-valued logic devices. The second school deals with the use of multi-valued algebras in the analysis of classical two valued logic hardware. Our work falls into this second category. We have carried out some of our analysis using Galois field theory. From this we are able to study certain of the cyclic state transition properties of sequential machines.

REFERENCES

1. P. R. Halmos, Finite Dimensional Vector Spaces, Van Nostrand, Princeton, New Jersey, 1958.
2. T. C. Bartee and D. I. Schneider, "Computations with Finite Fields," Information and Control, vol. 6, pp. 79-98, June 1963.
3. W. H. Kautz (ed.), Linear Sequential Switching Circuits, Holden-Day, San Francisco, 1965.
4. A. E. Ingham, "The Distribution of Prime Numbers," Cambridge Tracts in Mathematics and Mathematical Physics, no. 30, Cambridge University Press, London, 1932.
5. Y. Levendel, "Some Experiments and Problems in Fault Simulation," Department of Mathematics Report No. MATH-71, University of Negev, Israel, May 1974.
6. I. S. Reed, "A Class of Multiple-Error Correcting Code and the Decoding Scheme," IRE Trans. Inform. Theory, vol. IT-4, pp. 38-49, September 1954.
7. Y. Levendel and M. A. Breuer, "Application of Boolean Transform Theory to Test Generations," submitted to IEEE Trans. on Computers.
8. K. K. Saluja and S. M. Reddy, "Fault Detecting Test Sets of Reed Muller Canonic Networks," IEEE Computer Repository, R73-109, 1973.
9. Y. Levendel, "Algebraic Generation of Test and Synchronizing Sequences for Digital Circuits," Ph.D. Dissertation, University of Southern California, 1977.

OPTIMIZATION OF MULTIVALUED DECISION ALGORITHMS

A. Thayse, M. Davio, J.-P. Deschamps

MBLE Research Laboratory

$$\ell \wedge \bigwedge_{i=0}^{n-1} x_i^{(C_i)}, \ell \in \{0, 1, \dots, m-1\}, C_i \subseteq S \forall i$$

SUMMARY

One studies the possibility of evaluating the local values of a multivalued function by means of a specific type of algorithm called multivalued decision algorithm. The reason for this choice is its wide range of applications that encompasses both hardware and software problems. Possible applications of the multivalued decision algorithm are : combinatorial synthesis of discrete functions by means of multiplexers, sequential synthesis of discrete functions using multivalued ROM's and multiplexers, transformation and optimization of microprograms.

GENERAL DEFINITIONS AND NOTATIONS.

Let $\underline{x} = (x_{n-1}, \dots, x_1, x_0)$; a discrete function $f(\underline{x})$ is a mapping $f: S^n \rightarrow S$ with :

$$f: \{0, 1, \dots, m-1\}^n \rightarrow \{0, 1, \dots, m-1\}$$

The lattice exponentiation is defined as follows:

$$x_i^{(C_i)} = m-1 \text{ iff } x_i \in C_i \text{ with } C_i \subseteq \{0, 1, \dots, m-1\} = S \\ = 0 \text{ otherwise.}$$

The following operation symbols are used :

\vee or \bigvee stands for the disjunction, i.e.

$$a \vee b = \max(a, b); a, b \in S;$$

\wedge or absence of symbol stands for the conjunction, i.e. : $a \wedge b = \min(a, b)$;

A cube function is a discrete function of the form :

An implicant of a discrete function f is a cube function smaller than f ; a prime implicant of f is an implicant which is not smaller than any other implicant.

1. INTRODUCTION.

We consider first a discrete function of n variables $f(x_{n-1}, \dots, x_1, x_0)$ or, in short, $f(\underline{x})$. Various classical methods are available for realizing such a function by a combinational circuit : one could e.g. use a two-level circuit of AND-OR gates, NAND gates ... etc.

The synthesis goal generally consists in obtaining an acceptable compromise between design parameters such as e.g. the hardware cost and the response time. The essential features of the combinational synthesis are high speed and high cost and, for specific applications, this compromise may happen to be unacceptable. Hence the need to investigate alternatives to the combinational approach.

In the present paper, we study the possibility of evaluating the local values of a discrete function by means of a specific type of algorithm, to be defined formally below, and called *multivalued decision algorithm*. The reason of this choice is its wide range of applications, that encompasses both hardware and software problems. This point will be discussed with more details in what follows.

Definition 1. A multivalued decision algorithm is a loop free algorithm containing only a finite number of labelled instructions of the following type :

Let x_i be a discrete variable with values in $\{0, 1, \dots, m-1\}$; the instruction labelled N_j contains m statements labelled e.g. from 0 to $m-1$ and one variable $x_i \in \underline{x}$. It is defined as follows:
 N_j : If the variable x_i takes the value k_i go to the statement of N_j labelled k_i where :

$$(a) \quad k_i \in \{0, 1, \dots, m-1\}$$

- (b) the involved m statements are :
 either "go to label N_k " or "the value of
 the function f is h " with $h \in \{0, 1, \dots, m-1\}$.
 The latter statements are called *terminal*
statements.

The above kind of instruction corresponds to the SWITCH instruction in Algol and to the CASE instruction in Pascal respectively ; moreover if x_i is a binary variable these instructions reduce to the instruction *if - then - else*, i.e. :

N_j : *if* (condition x_i) *then* (statement 0 for $x_i=0$)
else (statement 1 for $x_i=1$).

If one has to evaluate a Boolean function, the multivalued decision algorithm will be called *binary decision algorithm*. We furthermore require the existence of a unique initial instruction that we denote N_1 . In what follows, the instructions of the type (1) will be unambiguously represented by the $(m+2)$ -tuple :

$$(j; x_i; s_0, s_1, \dots, s_{m-1}) \quad (2)$$

where j is the label, x_i is the discrete variable whose value is to be tested and where s_0, s_1, \dots, s_{m-1} are the m statements.

A multivalued decision algorithm is displayed as a labelled graph : we associate a node marked (j, x_i) with each instruction (2) and a node marked h ($h \in \{0, 1, \dots, m-1\}$) with each terminal statement. From a node (j, x_i) corresponding to the instruction (2) we draw (at most) m edges marked $0, 1, \dots, m-1$ to the nodes corresponding to the statements s_0, s_1, \dots, s_{m-1} respectively, if these statements are such that e.g. $s_k = s_\ell = \dots = s_m$ we draw from (j, x_i) an edge labelled k, ℓ, \dots, m to the node corresponding to the statement s_k .

The *in-degree* of a node is the number of edges drawn to it ; a multivalued decision algorithm will be called a multivalued decision tree if the in-degree of all its nodes (j, x_i) is at most 1.

Example 1. The multivalued decision algorithm.

N	x	0	1	2
1	x_2	N_2	N_3	N_2
2	x_1	N_4	0	0
3	x_1	N_4	N_4	2
4	x_0	0	1	2

is represented graphically in figure 1. An easy computation shows that this algorithm defines the discrete function $f(x_2, x_1, x_0)$:

$$f : \{0, 1, 2\}^3 \rightarrow \{0, 1, 2\}$$

the Karnaugh map of which is given in figure 2.

It is clear from the above example that every multivalued decision algorithm completely defines a unique discrete function. Conversely, given a discrete function, there exist numbers of multivalued decision algorithms that compute that function. That observation immediately raises various optimization problems. In order to define appropriate optimization criteria, we first briefly review some possible applications of the problem at hand.

a) Combinatorial synthesis of a discrete function.

The general problem is to synthesize a discrete function with a minimum number of combinational multiplexers. Let us recall that a multiplexer is a combinational network made up of m data inputs a_i and one control input x realizing the following output function z :

$$z = \bigvee_{i=0}^{m-1} a_i x^{(i)} \quad (3)$$

It is clear that to each multivalued decision algorithm we may associate a combinational logic circuit made up of multiplexers. This point is illustrated in figure 3 for the function in example 1. In this application, the hardware cost is proportional to the number of multiplexers, i.e. to the number of instructions in the multivalued decision algorithm, while the response time is proportional to the logical depth, i.e. to the maximum number of multiplexers encountered when passing from one input to the output. In the graph model, the logical depth is represented by the maximum distance between the initial node and the terminal node.

b) Sequential synthesis of discrete functions.

The problem of investigating binary decision algorithms seems to have first been tackled by C. Lee¹. More recently R. Boute² observed that, as any algorithm, a binary decision algorithm may be implemented as a sequential machine and, more precisely by microprogrammed-like architectures. In particular as the automaton associated with a binary decision is a condition automaton, in the sense of Ito³, it is obviously possible to subdivide the control store word in four fields : a variable identification field (V.I.F.), two address fields and an output field (see for instance Davio and Thayse⁴). The generalization of this synthesis method to the case of multivalued decision algorithms is straightforward. Let us illustrate the method by the function of example 1. First, let us slightly modify the algorithm in order to avoid the presence of two different types of statements :

N	x	0	1	2	O.A	O.V
1	2	N ₂	N ₃	N ₂	0	-
2	1	N ₄	N ₅	N ₅	0	-
3	1	N ₄	N ₄	N ₇	0	-
4	0	N ₅	N ₆	N ₇	0	-
5	-	N ₅	N ₅	N ₅	1	0
6	-	N ₆	N ₆	N ₆	1	1
7	-	N ₇	N ₇	N ₇	1	2

The presence of a "1" in the column O.A (Output Available) indicates that the computation is completed and that the value taken by f at point (x_2, x_1, x_0) is available in column O.V (Output Value). The implementation of this algorithm by means of ternary ROM's address decoders, multiplexers and registers is only a matter of encoding of the labels N_j , $j=1$ to 7. The following encoding has been chosen in figure 4 :

N ₁	N ₂	N ₃	N ₄	N ₅	N ₆	N ₇
00	01	02	10	20	21	22

The working of the circuit may be briefly explained :

- the address presently contained in the Instruction Address Register selects one of the rows of the Control ROM by means of the Address Decoder.
- The Variable Identification Field (V.I.F.) of the selected row selects one of the inputs x_0 , x_1 and x_2 by means of the multiplexer 2.
- The value of the selected input variable selects one of the three address fields and determines the next instruction address by means of the multiplexers b and c ; this new address is stored in the Instruction Address Register.
- Once the computation is completed, a "1" appears on the Output Available wire and $f(x_2, x_1, x_0)$ is available on the Output Value wire.

(Notice that the bars stand for don't care conditions).

In the general case of a multivalued decision algorithm computing a n -variable m -valued function in p steps, a row of the control memory must contain the following informations :

V.I.F.	Address 1	Address m	O.A QV
(1 out of n)	(1 out of p)	(1 out of p)	

If we use a m -ary ROM for storing those informations, the number of cells per row is

$$\lceil \log_m n \rceil + m \lceil \log_m p \rceil + 2.$$

Hence, the total number of cells in the control store must be greater than or equal to

$$p(2 + \lceil \log_m n \rceil + m \lceil \log_m p \rceil)$$

The number p of instructions is again the most important factor in the hardware cost evaluation.

An important feature of this type of synthesis is that the response time is data dependent. It becomes thus interesting to have an estimation of the average processing time. If in our running example, we assume that all the variables take the values 0,1 and 2 with probabilities equal to 1/3, the average processing time expressed in clock periods is 2.4 since 12 of the 27 cases are processed in 3 time units and 15 in 2 time units.

c) Application to microprogrammed systems.

It has already been mentioned that binary decision algorithms reduce to condition automata. Clearly enough, the problem of obtaining a binary decision algorithm for computing a Boolean function plays an essential role in transforming an arbitrary microprogram with the various consequence this fact may imply on the architecture of the control part of the system. (see e.g. Davio⁵). The sequential evaluation of discrete functions thus appears as a typical case of process sequentialization and thus of discussion of cost versus time tradeoffs.

In conclusion to the above discussion, we note that three possible optimization criteria have been defined :

- (a) the number of instructions in a multivalued decision algorithm $P(f)$;
- (b) the maximum processing time $\tau_M(f)$;
- (c) the average processing time $\bar{\tau}(f)$.

The problem of minimizing the average processing time (of Boolean functions) has recently been tackled by Breitbart and Reiter⁶ (unate functions) and Perl and Breitbart⁷. Note that the optimization of binary decision algorithms is related to that of transforming decision tables into computer programs and to the problem of organizing files as binary search trees. Survey paper on these topics have been presented by Poon⁸ and Nievergelt¹⁰ respectively.

The present paper attempts to present for the first time results and applications for the multivalued decision algorithm (it is understood that all the results in the literature quoted hereabove are exclusively concerned with binary decision algorithm). Moreover these results give rise to new and very competitive methods for Boolean functions.

The algorithms presented in this paper are based on the concepts of Twin-term and of weighted Twin-term which are introduced in sections 2 and 3 respectively. Further on these concepts will be referred to as T-term and as weighted T-term respectively. The concept of T-term is used in section 2 for the minimization of the maximum processing time in multivalued trees. The concept of weighted T-terms is used in section 3 for the minimization of the average processing time and of the number of instructions in a multivalued decision tree. Finally section 4 presents a method for reducing the number of instructions in multi-

valued decision algorithms.

2. THE T-TERMS AND THE OPTIMIZATION OF THE MAXIMUM PROCESSING TIME IN DECISION TREES.

We first define a generalized version of the consensus operation.

Definition 2.

Let $C = \bigwedge_{i=0}^{n-1} x_i^{(C_i)}$ and $C' = \bigwedge_{i=0}^{n-1} x_i^{(C'_i)}$ be two

cubes. The consensus of C and C' with respect to the variable x_k or k -th consensus of C and C'

is the cube denoted $C * C'$ and defined by :

$$C * C' = \bigwedge_k x_k^{(C_k \cup C'_k)} \bigwedge_{i \neq k} x_i^{(C_i \cap C'_i)} \quad (4)$$

It could easily be verified (see Davio and Bioul¹¹) that the k -th consensus is a commutative and associative operation on the set of cubes and that any consensus of two cubes is included in their disjunction.

Definition 3.

(a) The T -terms of order 0 or T^0 -terms of the discrete function $f(x)$ will be denoted $T_i^0(f)$ with i a counting index ; they are all the cubes covering the subdomains where the function f takes uniformly the same value $h(h \in \{0,1,\dots,m-1\})$.

(b) The T -terms of order $q(q \geq 1)$ or T^q -terms of the discrete function $f(x)$ will be denoted

$T_i^q(f)$ and are defined as follows :

$$T_i^q(f) = T_j^{q-1}(f) * \bigwedge_k T_{\ell_0}^{p_0}(f) * \bigwedge_k T_{\ell_1}^{p_1}(f) * \dots * \bigwedge_k T_{\ell_{s-1}}^{p_{s-1}}(f),$$

at most m cubes, $p_e \leq q-1$, $0 \leq e \leq m-1$

is a T^q -term if :

(a) The $s+1$ exponents C_e ($0 \leq e \leq s+1 \leq m$) of $x_k^{(C_e)}$ that appear in the $s+1$ cubes $T(f)$ of the right member of the above relation constitute a partition of the set $\{0,1,\dots,m-1\}$.

(b) $T_i^q(f) \neq T_{\ell}^r(f) \forall r \leq q-1$ and $\forall \ell$.

Stated otherwise, the consensus between a T^{q-1} -term

and at most $(m-1)T^e$ -terms ($p_e \leq q-1 \forall e$) produces

a T^q -term iff (a) the result of the consensus operation is independent of x_k and (b) the result is not a T^r -term ($r \leq q-1$).

Definition 4.

A T -term of order q will said to be a prime T^q -term iff it is not contained in another T^q -term.

Example 1 (Continued)

Consider again the example 1 together with the table 1 ; the prime T^0 -terms of the function of figure 2 are numbered from 1 to 8. Its prime

T^1 -terms, prime T^2 -terms and prime T^3 -terms are numbered from 9 to 17, from 18 to 23 and from 24 to 26 respectively. The following information is gathered in the column "consensus operation" : e.g. for the term numbered 17 one has : $(x_0:6,5,2)$ which means that the term 17 has been obtained by performing the consensus operation between the terms numbered 6,5 and 2 with respect to the variable x_0 . The letters $x_0^{(0)}$, $x_0^{(1)}$ and $x_0^{(2)}$ are present in the terms 6,5 and 2 respectively. In the same way the information $(x_2:14,11,14)$ for the term 21 means that this last has been obtained by consensus operation between the terms 11 and 14 and with respect to the variable x_2 ; the letters $x_2^{(0,2)}$ and $x_2^{(1)}$ are present in the terms 14 and 11 respectively.

q	N	T^q -terms	Consensus operation	Weights
0	1	$x_0^{(2)} x_1^{(0)}$	-	0,0
0	2	$x_0^{(2)} x_2^{(1)}$	-	0,0
0	3	$x_1^{(2)} x_2^{(1)}$	-	0,0
0	4	$x_0^{(1)} x_1^{(0)}$	-	0,0
0	5	$x_0^{(1)} x_2^{(1)} x_1^{(0,1)}$	-	0,0
0	6	$x_0^{(0)} x_1^{(0,1)}$	-	0,0
0	7	$x_1^{(1,2)} x_2^{(0,2)}$	-	0,0
0	8	$x_0^{(0)} x_2^{(0,2)}$	-	0,0
1	9	$x_1^{(0)}$	$x_0 : 6,4,1$	1,1
1	10	$x_0^{(2)} x_1^{(1,2)}$	$x_2 : 7,2,7$	1,1
1	11	$x_0^{(1)} x_2^{(1)}$	$x_1 : 5,5,3$	1,1
1	12	$x_0^{(0)} x_2^{(1)}$	$x_1 : 6,6,3$	1,1
1	13	$x_1^{(2)}$	$x_2 : 7,3,7$	1,1
1	14	$x_0^{(1)} x_2^{(0,2)}$	$x_1 : 4,7,7$	1,1
1	15	$x_0^{(1)} x_2^{(1)}$	$x_1 : 5,5,3$	1,1
1	16	$x_0^{(1)} x_1^{(1)}$	$x_2 : 7,5,7$	1,1

1	17	$x_1^{(0,1)} x_2^{(1)}$	$x_0 : 6, 5, 2$	$1, 1$
2	18	$x_1^{(1)}$	$x_2 : 7, 17, 7$	$2, \frac{4}{3}$
2	19	$x_0^{(0)}$	$x_2 : 8, 12, 8$	$2, \frac{4}{3}$
2	20	$x_0^{(2)}$	$x_1 : 1, 10, 10$	$2, \frac{5}{3}$
2	21	$x_0^{(1)}$	$x_2 : 14, 11, 14$	$3, 2$
2	22	$x_2^{(0,2)}$	$x_1 : 9, 7, 7$	$2, \frac{4}{3}$
2	23	$x_2^{(1)}$	$x_1 : 17, 17, 3$	$2, \frac{5}{3}$
3	24	$m-1$	$x_1 : 9, 18, 13$	$5, \frac{19}{9}$
3	25	$m-1$	$x_0 : 19, 21, 20$	$8, \frac{8}{3}$
3	26	$m-1$	$x_2 : 22, 23, 22$	$5, \frac{22}{9}$

Table I.

The algorithm, for optimizing the maximum processing time in decision trees, that will be developed is grounded on the following observation :

to each T^q -term $\bigwedge_i x_i^{(e_i)}$ that has been obtained by the consensus operation (see the notation of table I) :

$(x_k : N_{j_1}, N_{j_2}, \dots, N_{j_\ell})$ can be associated a multivalued decision tree having a maximum processing time of q , starting with x_k as decision variable and describing that part of f limited to the subdomain characterized by the equation :

$$\bigwedge_i x_i^{(e_i)} = m-1 \quad (5)$$

Indeed, consider this T^q -term as the root-node of the decision tree ; from this node are issued at most m branches (according to the number of the different N_{j_e} 's) associated to the values $0, 1, \dots$,

$m-1$ of the decision variable x_k and leading to the nodes numbered $N_{j_1}, N_{j_2}, \dots, N_{j_\ell}$ respectively. These nodes are in turn each associated with T^p -terms ($e \leq m, 0 \leq p_e \leq q-1 \forall e$); these T^p -terms are each considered as nodes of the multivalued decision tree and the process is continued iteratively until arriving to the terminal nodes which correspond to the T^0 -terms. Clearly the multivalued decision tree so constructed has a maximum processing time q and computes that part of f corresponding to the domain characterized by the equation (5).

Based on the above observations the following algorithm may now be stated.

Algorithm 1.

Starting from the prime T^0 -terms, compute the prime T^q -terms until a level r has been obtained such that at least one T^q -term is equal to $m-1$. To this T^q -term corresponds a multivalued decision tree having a maximum processing time of value q ; this value is minimal since the T^q -term is prime.

Example 1 (Continued)

Consider the table I ; to the prime T^3 -terms 24, 25 and 26 correspond the optimal decision trees of figures 5(b), (a) and (c) respectively. They have all a maximal processing time of 3.

3. THE WEIGHTED T-TERMS AND THE OPTIMIZATION OF THE NUMBER OF INSTRUCTIONS AND OF THE AVERAGE PROCESSING TIME

It has been seen that the concept of prime T-term was the adequate mathematical tool for the optimization of maximal processing times in multivalued decision trees. The concept of weighted T-term that will be introduced herebelow will be used in the optimization of the number of instructions and of the average processing time in multivalued decision trees.

Definition 5.

A weighted T-term is a pair

$$\{\alpha, c(\underline{x})\}$$

where α is a nonnegative rational number called the weight of the T-term and where $c(\underline{x})$ is a cube. The weighted T-terms of the discrete function $f(\underline{x})$ are defined iteratively as follows :

(a) The weighted T-term of order 0 or weighted T^0 -terms of the discrete function $f(\underline{x})$ are the pairs

$$\{m_0, \mathcal{T}_i^0(f)\} = \{0, T_i^0(f)\} \quad (6)$$

(b) The weighted T-terms of order q or weighted T^q -terms of the discrete function $f(\underline{x})$ are the pairs

$$\{m_q, \mathcal{T}_i^q(f)\}.$$

They are defined as follows : let

$$\{m_{q-1}, \mathcal{T}_j^{q-1}(f)\}, \{m_\ell^p, \mathcal{T}_\ell^p(f)\}, \dots, \{m_{s-1}^{p_{s-1}}, \mathcal{T}_{s-1}^{p_{s-1}}(f)\}$$

$$p_e \leq q-1 \forall e, s \leq m-1$$

be at most m weighted T-terms. Then

$$\{m_q, \mathcal{E}_i^q(f)\} = \{\phi(m_{q-1}, m_{\ell_0}, \dots, m_{\ell_{s-1}})\},$$

$$\mathcal{E}_j^{q-1}(f) * \mathcal{E}_{\ell_0}^{p_0}(f) * \dots * \mathcal{E}_{\ell_{s-1}}^{p_{s-1}}(f) \quad (7)$$

(with ϕ a function to be defined further on) is a weighted T^q -term if :

(a) The condition (a) of definition 3 is satisfied;

(b) either $\mathcal{E}_i^q(f) \neq \mathcal{E}_\ell^r(f)$, $r \leq q-1$ and $\forall \ell$

or $\mathcal{E}_i^q(f) = \mathcal{E}_\ell^r(f)$ for some $r \leq q-1$ and some ℓ

and $m_r > m_q$ for the weighted T^r -term

$$\{m_r, \mathcal{E}_\ell^r(f)\}.$$

Definition 6.

A *prime weighted T^q -term*, is a weighted T^q -term $\{m, c(\underline{x})\}$ such that for any other weighted T^q -term $\{m', c'(\underline{x})\}$ one has either

$$c(\underline{x}) \subseteq c'(\underline{x})$$

or if $c(\underline{x}) \subseteq c'(\underline{x})$ then $m < m'$.

Two types of functions $\phi(m_{q-1}, m_{\ell_0}, \dots, m_{\ell_{s-1}})$ will now be defined :

$$\phi_0(m_{q-1}, m_{\ell_0}, \dots, m_{\ell_{s-1}}) = m_{q-1} + m_{\ell_0} + \dots + m_{\ell_{s-1}} + 1 \quad (8)$$

$$\phi_1(m_{q-1}, m_{\ell_0}, \dots, m_{\ell_{s-1}}) = \frac{1}{s+1} (m_{q-1} + m_{\ell_0} + \dots + m_{\ell_{s-1}}) + 1 \quad (9)$$

Example 1 (continued)

The weights of the T^q -terms of table I and with respect to the laws ϕ_0 and ϕ_1 have been gathered in the column "weight".

The algorithm quoted in this section is grounded on the following observation ; to each

weighted T^q -term $\bigwedge_i x_i^{(e_i)}$ with weights m_0 and m_1 (with respect to the laws ϕ_0 and ϕ_1 respectively)

can be associated a multivalued decision tree having a maximum processing time q , a number m_0 of instructions, an average processing time m_1 and describing that part of f limited to the subdomain characterized by the equation

$$\bigwedge_i x_i^{(e_i)} = m-1$$

Indeed, it can easily be verified that the laws

ϕ_0 and ϕ_1 , compute the number of instructions and the average processing time of the considered multivalued decision tree respectively.

Algorithm 2.

Compute the prime weighted T -terms of the function f ; from the list of prime weighted T -terms equal to $m-1$ select those having a minimal weight m_0 (resp. m_1). To those weighted T -terms correspond multivalued decision trees having a minimum number of m_0 instructions (resp. a minimum average processing time of value m_1).

Example 1 (continued)

Consider the multivalued decision trees corresponding to the T^3 -terms 24, 25 and 26 of table I ; they have a number of instructions of 5, 8 and 5 respectively and an average processing time of 19/9, 8/3 and 22/9 respectively. This can also be verified by considering the three networks of figure 5.

4. REDUCTION OF THE NUMBER OF INSTRUCTIONS IN MULTIVALUED DECISION ALGORITHMS

Until now we have only considered minimization problems in the restricted frame of multivalued decision trees. Minimizing the number of instructions in multivalued decision algorithms is a much more complex problem.

We have seen in the preceding sections that minimization problems in decision trees may be solved (at least formally) by using the concept of T^q -term possibly associated with a weight; each T^q -term covers a subdomain of the function domain as characterized by the equation (5). If one wants to deal with the minimization of the number of instructions in decision algorithms, one has not only to consider the function subdomains but also the subfunctions that are realized in these subdomains : indeed, the minimization of the number of instructions requests that instructions should be merged, i.e. that identical subfunctions should be recognized. This will appear more clearly in the course of example 1 that will be continued further on in this section.

To each T^q -term will be associated a discrete function in the following way.

Definition 7.

(a) To a T^0 -term : $\bigwedge_i x_i^{(e_i)}$ which covers a subdomain where f takes uniformly the value h will be associated the function h . The pairs $\{T^0\text{-term, associated function}\}$ will be written

$$\{\bigwedge_i x_i^{(e_i)}, h\}$$

(b) Let $\{T_j^{q-1}(f), g_j\}$, $\{T_{\ell_0}^{p_0}(f), g_{\ell_0}\}$, ... ,

$\{T_{\ell_{s-1}}^{p_{s-1}}(f), g_{\ell_{s-1}}\}$ be the T -terms with their

associated function ; if :

$$T_i^q(f) = T_j^{q-1}(f) *_{k_0} T_{\ell_0}^{p_0}(f) *_{k_1} \dots *_{k_{s-1}} T_{\ell_{s-1}}^{p_{s-1}}(f)$$

is a T^q -term and if one assumes that $x_k^{(C_j)}$ is present in $T_j^{q-1}(f)$ and $x_k^{(C_e)}$ is present in $T_{\ell_e}^{p_e}(f)$

$\forall e$, the function associated to $T_i^q(f)$ is :

$$g_i = g_j^{(C_j)} \vee g_{\ell_0}^{(C_{\ell_0})} \vee \dots \vee g_{\ell_{s-1}}^{(C_{\ell_{s-1}})} \quad (10)$$

The reduction of the number of instructions in multivalued decision algorithms will be obtained by *merging* some instructions in these trees ; the merging of instructions will be defined as follows.
Consider two instructions with their associated function, i.e. :

instruction N_j : $\{T_j^q(f), g_j\}$

instruction N_i : $\{T_i^p(f), g_i\}$

Definition 8

Two instructions N_j and N_i may be merged if and only if $g_j = g_i$; the merged instructions N_{ij} is then :

$$\text{instruction } N_{ij} : \{T_j^q(f) \vee T_i^p(f), g_i\} \quad (11)$$

The merging of more than two instructions may be performed iteratively. The following theorem constitutes a quite straightforward consequence of the above definition 8.

Theorem.

In a multivalued decision algorithm, the merging of instructions transforms the primitive algorithm in an equivalent one (that is an algorithm computing the same discrete function) having a smaller number of instructions.

It should also be noted that the merged instruction N_{ij} represented in (11) describes that part of f limited to the subdomain characterized by the equation

$$T_j^q(f) \vee T_i^p(f) = m-1 \quad (12)$$

It could evidently be possible to build straightforward algorithms for the minimization of the number of instructions in multivalued decision algorithm by using instead of T^q -terms or of weighted T^q -terms, the pair

$\{T^q\text{-term, associated discrete function}\}$. *Prime pairs* could then be defined in the same way as prime T^q -terms and as prime weighted T^q -terms. However, the number of pairs that must be taken into account renders algorithms similar as those developed in sections 2 and 3 practically unusable.

Another approach that might be used to reduce (but not to minimize) the number of instructions in decision algorithms is e.g. to evaluate systematically all the discrete functions associated with prime T^q -terms or with prime weighted T^q -terms. Either the algorithm of section 2, or the algorithm of section 3 is then used for satisfying the corresponding minimization criterion and the associated discrete functions are then used in an "a posteriori" treatment in order to reduce the number of instructions. It can easily be verified that the merging of instructions does not affect the minimal character of the maximal processing time or of the average processing time.

Example 1 (continued)

Consider the functions associated with the prime T^q -terms ; Since the functions associated with the instructions 9 and 17 are the same, these instructions may be merged and the network of figure 5(c) (with five multiplexers) is transformed into that of figure 3 having only four multiplexers.

REFERENCES

- 1 C. Lee, Representation of switching circuits by binary decision programs, Bell System T.J., 38, pp. 985-999, July 1959.
- 2 R. Boute, The binary decision machine as programmable controller, Euromicro Newsletter, 2, pp. 16-22, January 1976.
- 3 T. Ito, A theory of formal microprograms, Sig-micro Newsletter, 4, pp. 5-17, April 1973.
- 4 M. Davio and A. Thayse, Sequential evaluation of Boolean functions, MBLE Internal Report R341, March 1977.
- 5 M. Davio, Hardware implementation of algorithmic computations, MBLE Internal Report R333, July 1976.
- 6 Y. Breitbart and A. Reiter, Algorithms for fast evaluation of Boolean expressions, Acta Informatica, 4, pp. 107-117, 1975.
- 7 J. Perl and Y. Breitbart, Optimal sequential evaluation trees for Boolean functions, Information Sciences, 11, pp. 1-12, 1976.
- 8 U. Pooch, Translation of decision tables, ACM Computing surveys, 6, pp. 125-151, June 1974.
- 9 P. Spira, On the time necessary to compute switching functions, IEEEETC, C-20, pp. 104-105, January 1971.
- 10 J. Nievergelt, Binary search trees and file organization, ACM Computing surveys, 6, pp. 195-206, September 1974.
- 11 M. Davio and G. Bioul, Representation of lattice functions, Philips Research Reports, 25, pp. 370-388, October 1970.

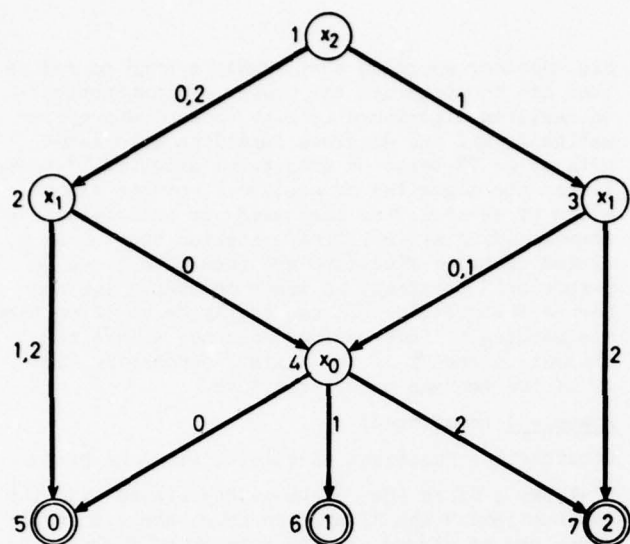


Figure 1 : decision algorithm

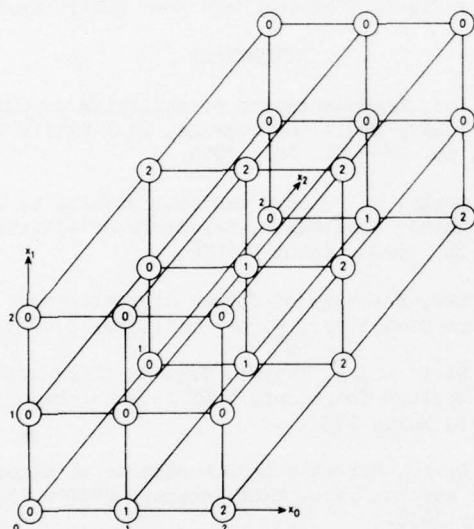


Figure 2 : value table

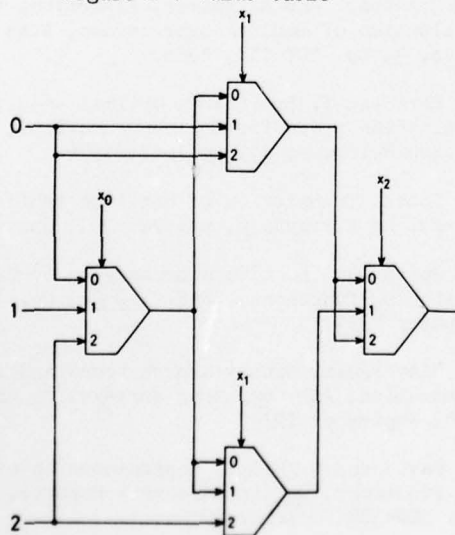


Figure 3 : multiplexer network

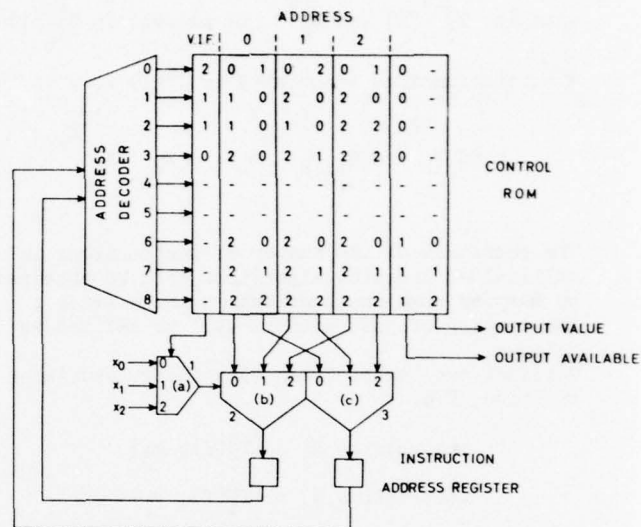


Figure 4 : sequential realization

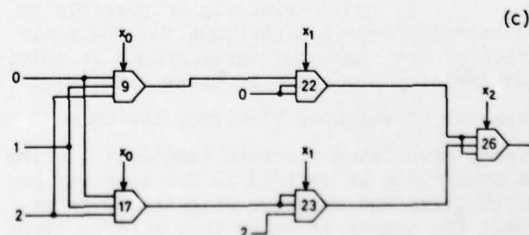
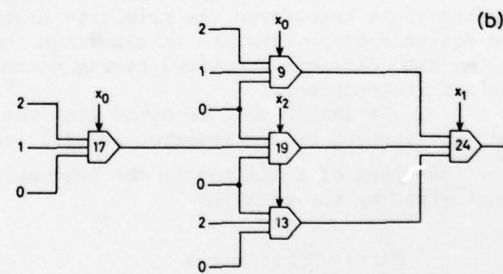
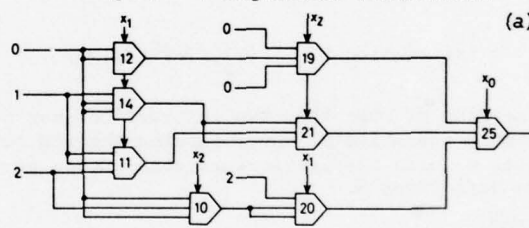


Figure 5 : optimal networks.

PARALLEL AND SERIAL DECOMPOSITIONS OF MULTI-VALUED SEQUENTIAL MACHINES*

T.C. Yang+
Department of Electrical & Power Engineering
National Tsing Hua University, Taiwan
Republic of China

A.S. Wojcik
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois

Abstract

The decomposition of a multi-valued sequential machine into an interconnection of two submachines with all feedback paths internal to the submachines is discussed. Different models for the interconnection of component machines along with the necessary and sufficient conditions for both serial and parallel decompositions are defined. Uni-covered state assignment is also defined. Examples are given to illustrate the decomposition procedures.

I. Introduction

One problem which has received considerable attention, at least in the binary case, is that of decomposing a sequential machine into an interconnection of two or more submachines with all feedback paths internal to the submachines. The engineering advantages of breaking down the overall specification of a large machine into several small machines are rather obvious: They may lead to circuits that are easier to understand, diagnose, maintain, and manufacture; and the overall component count is also likely to be reduced because in several types of decomposition, the number of states in the large machine is the product of the numbers of states of the small machines. In the binary case, Hartmanis and Stearns (1-4) developed the basic analysis for decomposition. Tan, Menon, and Friedman (5) extended the theories of reduced dependence and structural decomposition developed by Hartmanis and Stearns for synchronous sequential machines to asynchronous sequential machines. There are other reports on decomposition (6-12). Since the rapidly growing interest in multi-valued switching functions, it is important to study the decomposition

of multi-valued switching functions. Miller and Muzio (13-15) presented a technique for two-place decomposition of multi-valued combinational functions. This study is basically concerned with extending present methods for decomposing binary sequential machines, both synchronous and asynchronous, to methods for decomposition multi-valued sequential machines. Several procedures for defining and analyzing models for the interconnection of multi-valued sequential machines and for determining when and how a multi-valued sequential machine can be decomposed into a series or parallel connection of submachines are defined. Ternary machines are used in the examples only because they are, by far, the simplest to understand. All the methods to be presented are directly applicable (with no modification or just straightforward extensions) to arbitrary multi-valued systems.

II. Basic Concepts of Machine Structure Theory

A partition on a set of states of a machine is a grouping of the states into disjoint subsets, called blocks, such that every state belongs to exactly one block. By $s_1 = s_2(\pi)$, we mean that s_1 and s_2 are in the same block of partition π . The trivial partition with just one state in each block is called the ϕ -partition; whereas the partition with all states in one block is called the I-partition.

The ordered pair of partitions (π_1, π_2) defined on the set of states S is a partition pair for the machine M , denoted by $P(\pi_1, \pi_2)$, iff for each block B_i of π_1 and each input I_m , there exists a block B_j of π_2 such that $N(B_i, I_m) \subseteq B_j$, where N represents the next state function. A partition π defined on S is said to have the substitution property, denoted by $SP(\pi)$, if and only if $P(\pi, \pi)$. If a partition has the substitution property, then it is called a SP partition.

Let us denote the R logic values in a R -valued system as $0, 1, 2, \dots, r-1$. Let $\pi = (B_1, B_2, \dots, B_n)$ be an n -block partition on S . A R -valued y -variable covers π , denoted by τ_y , if y is assigned the values $0, 1, 2, \dots, a_1$ for states in

*This work was supported in part by the National Science Foundation under Grant MCS 74-01409A01.

+This research was performed at the Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois, and appears as part of Dr. Yang's Ph.D. thesis (18); some extensions of the thesis are also described in this paper.

$B_{i_1}; a_{i_1}+1, a_{i_1}+2, \dots, a_{i_2}$ for states in $B_{i_2}; \dots;$
 $a_{i_{n-1}}+1, a_{i_{n-1}}+2, \dots, r-1$ for states in B_{i_n} ,
 where $0 \leq a_{i_j} \leq r-1$ for $1 \leq j \leq n-1$ and B_{i_j} is a block
 in (B_1, B_2, \dots, B_n) for $1 \leq j \leq n$.

Since each state in S appears in one and only one block of the partition, the partition is said to be uncovered by a y -variable. If every y -variable in a state assignment covers a partition, then the assignment is called a uncovered state assignment. A uncovered state assignment in variables y_1, y_2, \dots, y_n on S is valid if $\pi_{i=1}^n$

$\tau_{y_i} = \emptyset$. If a valid uncovered state assignment is also a single transition time (STT) assignment, that is, all the state variables are allowed to change simultaneously without critical races, then it is called a uncovered single transition time assignment.

Based on Ungers' (16) theorem in the binary systems and Sheafor's (17) extension of that theorem to the multi-valued cases, we obtain the following theorem:

Theorem: A row assignment is a valid uncovered STT assignment iff, for every pair of transitions $s_i \rightarrow s_j$ and $s_m \rightarrow s_n$ that appear in the same column and such that $s_j \neq s_n$, the associated dichotomy $(s_i s_j, s_m s_n)$ is covered by at least one y -variable of that assignment.

In general, a valid uncovered state assignment does not always exist for an arbitrary machine; it is therefore necessary to extend the concept of the partition to the concept of a set system. A set system π on a set of states, S , is a collection of subsets B_1, B_2, \dots, B_i of S such that $U_i B_i = S$ and $B_i \not\subseteq B_j$ for $i \neq j$. A partition is a special case of a set system. The properties of set systems are basically similar to that of partitions. A state assignment obtained from a set system is called a multi-covered state assignment. Refer to (10) for a detailed discussion.

In this study we assume that delay elements are used as memory in the sequential machines. We also assume that we are not directly concerned with the output of the machine, but are primarily interested in the properties of the state transitions. Therefore, a sequential (state) machine is defined as $M=(S, I, N)$ where S, I, N represent a set of states, a set of inputs, and the next state (or transition) function, respectively.

Proofs of the theorems are not given in this paper. Interested readers can refer to (18) for details. In the following sections, we shall show how these concepts can be used to obtain a decomposition of a sequential machine. It is obvious that we are only interested in nontrivial

decomposition, that is, a decomposition in which all submachines have fewer states than the original machine. It should be pointed out again that all decomposition procedures to be discussed are applicable to arbitrary R -valued sequential machines.

III. Parallel Decomposition

Figure 1 shows a parallel decomposition of a machine M into component machines M_1 and M_2 . In this type of decomposition the next state of each machine depends only on the input and its own present state. The output of the component machine M will usually depend on the input and the present states of both M_1 and M_2 .

Definition A decomposition of a sequential state machine, denoted by $M=(S, I, N)$, into component sequential state machines, denoted by $M_1=(S_1, I_1, N_1)$ and $M_2=(S_2, I_2, N_2)$, is a parallel decomposition if

- a) $S = S_1 \times S_2$
- b) $I = I_1 \times I_2$
- c) $N[(s_1, s_2), (i_1, i_2)] = (N_1(s_1, i_1), N_2(s_2, i_2))$

where

$$s_1 \in S_1, s_2 \in S_2, i_1 \in I_1 \text{ and } i_2 \in I_2.$$

III.a Parallel Decomposition of Synchronous Machines

It follows from the previous definitions that the set system (partition) π_1 on the set of states, S , of machine M has the substitution property if and only if each input maps blocks of π_1 into blocks of π_1 . Since the operation of M determines unique block to block transformations on a SP set system π_1 , we can think of these blocks as the states of a new state machine M_{π_1} defined by π_1 and M . We can think of M_{π_1} as a machine which does only part of the computation performed by M , since it only keeps track of which block of π_1 contains the state of M .

To obtain an intuitive view of how these concepts can be used for machine decomposition, let us consider that we have two such machines, M_{π_1} and M_{π_2} , operating side by side. Operating separately, each machine does only part of the operation performed by M , since it only determines the block of π_1 or π_2 which contains the state of M . Operating jointly, M_{π_1} and M_{π_2} could determine the exact state of M if every block of π_1 has one

and only one state of M in common with every block of π_2 . Thus, the states of M_{π_1} and M_{π_2} uniquely determine a state of M , and hence when operating in parallel, these two machines compute the state transitions of M . These ideas are made precise in the following theorem which is essentially identical to that in the binary case (4, 19):

Theorem 3.1 A synchronous sequential machine M has a nontrivial parallel decomposition if and only if there exists two nontrivial SP set systems π_1 and π_2 on the set of states S of M such that $\pi_1 \cdot \pi_2 = \emptyset$.

The state tables of the component machine M_1 and M_2 of a parallel decomposition can be derived directly from π_1 , π_2 and M . The number of states in M_1 is equal to the number of blocks of π_1 . The number of states in M_2 is equal to the number of blocks of π_2 . The next state entries of M_j , $j=1,2$, are derived as follows: If a block B_i of π_j consists of states s_1, s_2, \dots, s_p of M , then $N_j(B_i, I_m)$ is the set of states $\{N_j(s_1, I_m), N_j(s_2, I_m), \dots, N_j(s_p, I_m)\}$. Since π_j has the substitution property, all the states in $N_j(B_i, I_m)$ will be in the same block B_k of π_j . Thus, if block B_i of π_j is represented by state s_{j_i} and B_k by s_{j_k} of M_j , then $N_j(s_{j_i}, I_m) = s_{j_k}$.

Example 1. Figure 2 gives the state table of M . Consider the set systems with SP, $\pi_1 = (12, 34, 56)$ and $\pi_2 = (135, 246)$. Since $\pi_1 \cdot \pi_2 = \emptyset$, these two partitions can be used to define a parallel decomposition of M into M_1 , as defined by variable $y_1 (\tau_{y_1} = \pi_1)$, and M_2 , as defined by variable $y_2 (\tau_{y_2} = \pi_2)$.

M_1 has three states, A, B, and C, corresponding to the three blocks of π_1 , 12, 34, and 56, respectively, and the three inputs I_0, I_1 , and I_2 of M . Let us illustrate the procedure for obtaining the next state entries of M_1 by deriving one entry, say

$N_1(B, I_1)$. Since $N_1(B, I_1) = N_1(34, I_1) = 12 \subseteq A$, $N_1(B, I_1) = A$. M_2 can be constructed in a similar way. The state tables of M_1 and M_2 are shown in Figures 3 and 4 respectively. Note that every state in M_1 and every state in M_2 will uniquely identify a state in M . For example, A of M_1 and b of M_2 identify state 2 of M because $A \cap b = (12) \cap (246) = 2$.

III.b. Parallel Decomposition of Asynchronous Machines

In this section, our attention is restricted to asynchronous machines operating in the fundamental mode, that is, the time between input changes is sufficient for the machine to reach the proper stable state. It is also assumed that only single input changes can occur, although these changes may be multiple-value changes. The conditions in multi-valued flow tables in which multiple-value input changes are allowed are based on the work done by Sheafor (17).

The conditions for the parallel decomposition of an asynchronous sequential machine are the same as those for the synchronous case. The following theorem is based on a similar result in the binary case (4, 19):

Theorem 3.2 An asynchronous sequential machine M has a nontrivial parallel decomposition if and only if there exists two nontrivial SP partitions π_1 and π_2 on the set of states S of M such that $\pi_1 \cdot \pi_2 = \emptyset$.

The flow tables of the component machines M_1 and M_2 can be directly derived from π_1 , π_2 , and M by using an approach similar to the one defined for the parallel decomposition of synchronous sequential machines. It should be noted that each of the component machines in the parallel decomposition must be realized with a valid asynchronous state assignment. The assignment to be discussed in this section is the single transition time state assignment, that is, all variables change simultaneously, and the order in which the variables change is irrelevant to the operation. It is obvious that if M_1 and M_2

are to be single transition time machines, then the machine M , formed by the parallel connection of M_1 and M_2 , is also a single transition time machine.

Example 2. The flow table of machine M is shown in Figure 5. Let us consider $\pi_1 = (123, 456)$ and $\pi_2 = (14, 25, 36)$. Since $\pi_1 \cdot \pi_2 = \emptyset$, these two SP partitions can be used to define a parallel decomposition of M into M_1 and M_2 as shown in Figures 6 and 7 respectively. Now, both M_1 and M_2 must be realized with valid asynchronous state assignments. Sheafor (17) designed a method, which is an extension of Tracey's STT assignment approach for the binary case, to generate a valid STT assignment for any multi-valued flow table. His method is applied to generate the state assignments for M_1 and M_2 as shown.

The parallel decomposition theorem is valid for a multicovered assignment if partitions in

Theorem 3.2 are generalized to set systems. An example is given in (18).

IV. Serial Decomposition

Figure 8 shows a serial decomposition of a machine M into component machines M_1 and M_2 . In this type of decomposition, all feedback paths are restricted to be internal to the component machines just as in the case of parallel decomposition. The next state of M_1 is independent of the state of M_2 , but the next state of M_2 depends on the present states of both M_1 and M_2 . The output depends on the input and the states of both machines.

Definition A decomposition of a sequential state machine $M=(S,I,N)$ into component sequential state machines, denoted by head machines $M_1=(S_1,I_1,N_1)$ and tail machine $M_2=(S_2,I_2,N_2)$, is a serial decomposition if

- $S = S_1 \times S_2$
- $I_1 = I$
- $I_2 = S_1 \times I_1$
- $N[(s_1, s_2), i] = (N_1(s_1, i), N_2(s_2, (s_1, i)))$

where

$$s_1 \in S_1, s_2 \in S_2 \text{ and } i \in I_1$$

IV.a Serial Decomposition of Synchronous Machines

Similar to the parallel case, serial decomposition of synchronous machines is based on the concepts of the substitution property and reduced dependency of the set of states. The following theorem is based on a similar result for the binary case (4,19):

Theorem 4.1 A synchronous sequential machine M has a nontrivial serial decomposition if and only if there exists a nontrivial SP set system π_1 on the set of states S of M .

Similar to the construction of M_1 and M_2 in the parallel decomposition, the head machine M_1 in the serial decomposition can be constructed according to π_1 . The tail machine M_2 can be constructed as follows: We assume that π_1 has ℓ blocks and the largest block has k states. Let π_2 be a k -block set system on S such that $\pi_1 \cdot \pi_2 = 0$. Such a π_2 can be constructed by labeling the states of each B_i of π_1 by $1, 2, \dots, N_i, N_i \leq k$, and then placing all states with the same label in one block of π_2 . For example, if $\pi_1 = (1234, 567, 89)$, then the labeling can be done as follows:

$$\pi_1 = (1234, 567, 89)$$

$$\begin{matrix} \dots & \dots & \dots \\ \dots & \dots & \dots \end{matrix}$$

$$\text{label } 1234 \quad 123 \quad 12$$

Now if states with the same label are grouped in a block, we can get $\pi_2 = (158, 269, 37, 4)$. The basic idea is that M_1 computes the block of π_1 which contains the state of M , and M_2 computes the corresponding block of π_2 . Since $\pi_1 \cdot \pi_2 = 0$, the serial connection of M_1 and M_2 computes the state of M . The state table of M_2 is derived directly from π_2 , M_1 and M in the following manner.

- The states of M_2 correspond to blocks of π_2 .
- The inputs to M_2 are the external inputs and the states of M_1 .
- For a state s_{2i} of M_2 and inputs (I_m, s_{1j}) , where $I_m \in I$ and s_{1j} is a state M_1 , the next state entry, $N_2(s_{2i}, (I_m, s_{1j}))$ is defined as follows: s_{1j} represents a block B_{1j} of π_1 , and s_{2i} represents a block B_{2i} of π_2 . Since $\pi_1 \cdot \pi_2 = 0$, we know that $B_{1j} \cap B_{2i} = s_p$ or 0 .
 - If $B_{1j} \cap B_{2i} = 0$, then combined state represents a don't care condition.
 - If $B_{1j} \cap B_{2i} = s_p$, the combined state of M_1 and M_2 represents a unique present state of M , and the combined next states of M_1 and M_2 must uniquely represent the next state of M . If the state s_p is contained in a single block B_{2p} of π_2 , then $N_2(s_{2i}, (I_m, s_{1j})) = s_{2p}$ which is the state representing the block B_{2p} of π_2 . If the states s_p is contained in more than one block of π_2 , then the next state for this entry can be any state of M_2 which represents s_p , since the combined next state for any of these possibilities will correctly represent s_p .

Example 3. For the state table shown in Figure 9, $SP(\pi_1) = (14, 25, 36)$ and $\pi_2 = (123, 456)$. These two set systems are used to obtain a serial decomposition of M . M_1 is defined by π_1 as shown in Figure 10; M_2 is defined by π_2 as shown in Figure 11. As an example, let us consider the next state entry $N_2(a, B_{10})$. Since $a = (123)$ and $B = (25)$, $a \cap B = 2$. In the state table of M , $N(2, 1_0) = 5$. Since $5 \in b$, we define $N_2(a, B_{10}) = b$.

If set systems are used to represent a multi-covered state assignment, that is there is at least

one state which appears in more than one but not all blocks in a set system, then the set systems π_1, π_2 which define a serial decomposition need not satisfy the condition $\pi_1 \cdot \pi_2 = \emptyset$. Refer to (18) for further discussion.

IV.b. Serial Decomposition of Asynchronous Machines

As in the case of the serial decomposition of synchronous machines, a necessary condition for the serial decomposition of an asynchronous machine M is that there exists a nontrivial SP set system π_1 on the set of states of M . The main difference between the serial decomposition of synchronous and asynchronous machines is in the tail machine M_2 .

Definition: A decomposition is a single transition time decomposition if the proper operation of the connection of the component machines is not dependent on the order in which the component machines change state.

Definition: A decomposition is a multiple transition time decomposition if the proper operation of the connection of the component machines is dependent on the order in which the component machines change state, and that order should be fixed by the insertion of suitable delays.

We first consider the single transition time decomposition. In order to obtain a single transition time serial decomposition, it is necessary that every pair of transitions $s_i \rightarrow s_j$ and $s_m \rightarrow s_n$ that appear in the same column and such that $s_j \neq s_n$, the associated dichotomy $(s_i s_j, s_m s_n)$ is covered by at least one state variable (17). Therefore, in the construction of the single transition time decomposition, we first make an STT state assignment for M_1 , and then use this state assignment as part of the assignment of M , and then assign additional state variables to cover the dichotomies of M not covered by the state variable of M_1 . The additional state variables obtained may be considered as the state variables of M_2 . A flow table for M_2 can then be found by forming a table whose rows correspond to the blocks of $\pi_{y_i \in \sigma} \pi_{y_i}$, where σ is the set of state variables of M_2 and π_{y_i} is the corresponding set system of variable y_i . The columns of the flow table for M_2 are the states of M_1 and the external inputs of M . Since the serial connection of M_1 and M_2 has the property that all variables that are required to change in a transition may be allowed to change simultaneously, the entries of the flow table of M_2 must be filled in so that M_2 -state changes can occur before or after M_1 -state changes. It is obvious that no nontrivial

STT serial decomposition exists if the state variables of M_1 do not cover any of the dichotomies of M . The following example will illustrate these concepts.

Example 4. Let us consider the flow table of M as shown in Figure 12, and the SP set system $\pi_1 = (12, 34, 567)$. The flow table of M_1 can be readily constructed as shown in Figure 13. The set of dichotomies, D_1 , that must be covered for a valid STT state assignment for M_1 is: $D_1 = \{(A, BC) (AB, C)\}$. A ternary state variable y_1 is used to cover D_1 by assigning values 0, 1, and 2 to states A, B, and C, respectively. Now let us consider M . The set of dichotomies, D , that must be covered for a valid STT state assignment for M is: In column I_0 - (12, 35), (12, 37), (12, 46), (35, 46) and (37, 46); In column I_1 - (14, 23), (14, 5), (14, 6), (14, 7), (23, 5), (23, 6), (23, 7), (5, 6), (5, 7), and (6, 7); In column I_2 - (13, 56), (13, 57), (23, 56), (23, 57), (34, 56), and (34, 57). All the dichotomies in M are covered by the state variable y_1 of M_1 except the following: (35, 46), (37, 46), (14, 23), (5, 6), (5, 7), (6, 7). These dichotomies can be combined to form the three dichotomies: (357, 46), (14, 23) and (5, 7). A ternary state variable y_2 can be used to cover them by assigning values 0, 1, and 2 for groups of states 146, 27, and 35, respectively.

Since $\pi_{y_2} = (146, 27, 35)$, M_2 has three states a, b, and c, corresponding to the three blocks of π_{y_2} , 146, 27, and 35, respectively. Now, let us fill in the next state entries in the flow of table of M_2 . As an example, let us start with M in (1, 1₁). M_1 is in (A, 1₁) and M_2 must be in (a, 1₁A). If the input now changes from I_1 to I_2 , $N_2(a, I_2A) = c$ because the final stable state according to M is 3, $3 \in c$ in M_2 , and therefore, $N_2(c, I_2A) = c$. Now consider M_1 . Since $N_1(A, I_2) = B$ and M_2 may see the M_1 -state change first, in order to make sure that the final stable state in M_2 is c, $N_2(a, I_2B) = N_2(c, I_2B) = c$. The complete flow table of M_2 is shown in Figure 14. Based on $\pi_{y_1} = (12, 34, 567)$ and $\pi_{y_2} = (146, 27, 35)$, the state assignment for M is obtained as shown in Figure 12.

Now let us consider the use of a multiple transition time decomposition. By fixing the order in which the component machines are allowed to change state, that is, by inserting delays into the component machines, it is possible to obtain simpler flow tables at the expense of speed. There are several models in which the order of operations on M_1 and M_2 can be fixed (12). The model to be used in this section is to insert delays between

M_1 and M_2 , as shown in Figure 15, and make certain that the delays are large enough to assume that M_2 will first see the change in the external input, will have time to stabilize before it sees a change of state in M_1 , and will then stay in the stabilized state.

The flow table of the head machine, M_1 , can be constructed from a SP set system π_1 as defined previously. The following steps define a procedure to obtain the flow table of M_2 .

1. Construct a flow table M'_2 such that
 - 1.A. M'_2 has a column corresponding to every combination of external input and internal state of M_1 .
 - 1.B. M'_2 has a row corresponding to every internal state of M .
 - 1.C. The next state entries in the flow table M'_2 can be determined as follows:
 - 1.C.a. Let the column I_k of M contain a transition $s_i \rightarrow s_j$ and let B_i and B_j be the state in M_1 corresponding to the blocks of π_1 that contain the states s_i and s_j , respectively. M'_2 is constructed such that $N'_2(s_i, (I_k, B_i)) = N'_2(s_j, (I_k, B_j)) = N'_2(s_j, (I_k, B_j)) = s_j$, and $N'_2(s_i, (I_k, B_j))$ is left unspecified because M_2 changes before M_1 . These entries are shown in Figure 16.
 - 1.C.b. Repeat step 1.C.a. until no further transitions exist. The remaining entries in M'_2 are left unspecified.
2. Reduce M'_2 . Any pair of states which are in the same block of π_1 cannot be merged because the combined states of M_1 and M_2 should uniquely identify a state in M . The reduced flow table M'_5 is the flow table of the tail machine, M_2 .

Note that in the above procedure, if M'_2 cannot be reduced, then M_2 and M have the same number of states and the SP set system π_1 yields a trivial serial decomposition. The following example will illustrate these concepts.

Example 5. Let us consider the flow table of the machine M shown in Figure 12. Since $SP(\pi_1) = (12, 34, 567)$, the flow table of M_1 can be constructed as shown in Figure 13. Now let us construct the flow table M'_2 . As an example, let us start with M in total state $(1, I_1)$, M_1 is in total

state (A, I_1) and M_2 must be in total state $(1, I_1A)$. If the input now changes from I_1 to I_2 , the final stable total state in M is $(3, I_2)$. Since $3 \in B$ and the corresponding stable total state in M_1 is (B, I_2) , $N'_2(1, I_2A) = N'_2(3, I_2A) = N'_2(3, I_2B) = 3$, and $N'_2(1, I_2B)$ is left unspecified. The complete flow table M'_2 is shown in Figure 17. The reduction can be done by using the concept of the pair chart (16) by crossing out those pairs that are in the same block of π_1 . The result so obtained is $(146, 235, 7)$ and the reduced flow table for M_2 is shown in Figure 18.

V. Conclusion

This paper has presented a number of systematic procedures for the parallel and serial decompositions of multi-valued synchronous and asynchronous sequential machines. Different models for the interconnection of component machines along with the necessary and sufficient conditions for various kinds of decompositions have been defined. Both single transition time operation and multiple transition time operation were considered for the serial decomposition case. A number of examples to illustrate these decomposition procedures have also been presented. One of the most important concepts developed in this study is the realization of the analogy between binary and multi-valued sequential machines. As proved in this study, many of the decomposition algorithms defined for the binary sequential machines were extendable to multi-valued sequential machines.

In the binary case, complex decompositions, that is, decomposing a sequential machine into three or more component machines, can often be decided upon in advance by a study of the SP lattice (4), that is, a structure that reveals all of SP partitions or set systems of a given table. It appears that a similar result can be expected for multi-valued cases. Since only the state behavior of a machine was considered in this work, further study is needed to consider the effect on the outputs due to decomposition.

VI. References

1. Hartmanis, J. 1961. On the Statement Assignment Problem for Sequential Machines, I. IRE Transactions on Electronics Computers, Vol. EC-10, No. 2: 157-165.
2. Stearns, R.E., and Hartmanis, J. 1961. On the State Assignment Problem for Sequential Machines, II. IRE Transactions on Electronic Computers, Vol. EC-10, No. 4: 593-603.
3. Hartmanis, J., and Stearns, R.E. 1964. Pair Algebra and its Application to Automata. Information and Control, Vol. 7, No. 4: 485-507.

4. Hartmanis, J., and Stearns, R.E. 1966. Algebraic Structure Theory of Sequential Machines. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
5. Tan, C.J., Menon, P.R., and Friedman, A.D. 1969. Structural Simplification and Decomposition of Asynchronous Sequential Circuits. IEEE Transactions on Computers, Vol.C-18, No.9:830-838.
6. Weiner, P., and Smith, E.J. 1967. Optimization of Reduced Dependencies for Synchronous Sequential Machines. IEEE Transactions on Computers, Vol.C-16, No.12:835-847.
7. Yoeli, M. 1961. The Cascade Decomposition of Sequential Machines. IRE Transactions on Electronic Computers, Vol.EC-10, No.4:586-592.
8. Jump, J.R. 1969. A Note on the Iterative Decomposition of Finite Automata. Information and Control, Vol.15, No.5:424-435.
9. Harlow, C., and Coates, C.L. 1967. On the Structure of Realization Using Flip-Flop Memory Elements. Information and Control, Vol.10, No.2:159-174.
10. Krohn, K.B., and Rhodes, J.L. 1965. Algebraic Theory of Machines, I. Prime Decomposition Theorem for Finite Semigroups and Machines. Trans. Amer. Math. Soc., Vol.116:450-464.
11. Zeiger, H.P. 1967. Cascade Synthesis of Finite State Machines. Information and Control, Vol.10, No.4:419-433.
12. Kinney, L.L. 1970. Decomposition of Asynchronous Sequential Switching Circuits. IEEE Transactions on Computers, Vol.C-19, No.6: 515-529.
13. Muzio, J.C., and Miller, D.M. 1973. Decomposition of Ternary Switching Functions. Conference Record of the 1973 International Symposium on Multiple-Valued Logic, Toronto, Canada.
14. Miller, D.M., and Muzio, J.C. 1976. Two-Place Decomposition and the Synthesis of Many-Valued Switching Circuits. Proceedings of the Sixth International Symposium on Multiple-Valued Logic, Logan, Utah.
15. Miller, D.M. 1976. Decomposition in Many-Valued Logic Design, Ph.D. Thesis, Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada.
16. Unger, S.H. 1969. Asynchronous Sequential Switching Circuits. John Wiley & Sons, Inc., New York, New York.
17. Sheafor, S.J. 1974. The Design of Multiple-Valued Asynchronous Sequential Circuits. Ph.D. Thesis, University of Illinois, Urbana, Illinois.
18. Yang, T.C. 1977. The Decomposition of Multi-Valued Sequential Machines. Report No.77-12, Department of Computer Science, Illinois Institute of Technology. Chicago, Illinois.
19. Friedman, A.D., and Menon, P.R. 1975. Theory and Design of Switching Circuits. Computer Science Press, Inc., Woodland Hills, California.

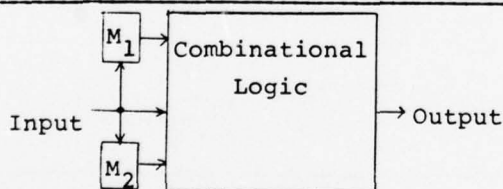


Fig.1. Parallel Decomposition

State	Input			State Assignment	
	I_0	I_1	I_2	Y_1	Y_2
1	4	5	2	0	0
2	3	6	1	0	1,2
3	6	1	4	1	0
4	5	2	3	1	1,2
5	2	3	6	2	0
6	1	4	5	2	1,2

Fig.2. State Table of M in Example 1

τ_{Y_1}	State	Input		
		I_0	I_1	I_2
12	A	B	C	A
34	B	C	A	B
56	C	A	B	C

Fig.3. State Table of M_1 in Example 1

τ_{Y_2}	State	Input		
		I_0	I_1	I_2
135	a	b	a	b
246	b	a	b	a

Fig.4. State Table of M_2 in Example 1

Fig. 11. State Table of M_2 in Example 3									
M_2 State		AI_0	AI_1	AI_2	BI_0	BI_1	BI_2	CI_0	CI_1
123	a	b	a	a	b	a	b	a	a
456	b	b	a	b	b	b	b	b	b

Fig. 10. State Table of M_1 in Example 3									
$SP(\Pi_1)$		State	I_0	I_1	I_2	Input			
14	A	A	A	A	B				
25	B	B	C	B	B				
36	C	C	C	C	B				

Fig. 9. State Table of M in Example 3									
State		Input	I_0	I_1	I_2	Y_1	Y_2		
1	4	1	2	0	0	0	0		
2	5	3	2	1	0	1	0		
3	6	3	2	2	0	2	0		
4	4	4	5	0	1,2	0	1,2		
5	5	6	5	1	1,2	0	1,2		
6	6	6	5	2	1,2	0	1,2		

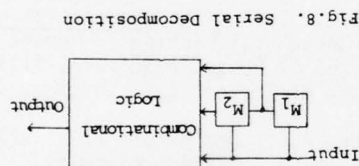


Fig. 7. Flow Table of M_2 in Example 2									
State		Input	I_0	I_1	I_2	Y_1	Y_2		
14	a	a	a	a	b	0	0		
25	b	b	c	b	b	1	1		
36	c	c	c	c	b	2	2		

Fig. 6. Flow Table of M_1 in Example 2									
State		Input	I_0	I_1	I_2	Y_1	Y_2		
123	A	B	A	A	A	0	0		
456	B	B	B	B	B	1,2	1,2		

Fig. 5. Flow Table of M in Example 2									
State		Input	I_0	I_1	I_2	Y_1	Y_2		
1	4	1	2	0	0	0	0		
2	5	3	2	1	0	1	0		
3	6	3	2	2	0	2	0		
4	4	4	5	0	1,2	0	1,2		
5	5	6	5	1	1,2	0	1,2		
6	6	6	5	2	1,2	0	1,2		

		Input		(Example 4) State Assignment		(Example 5) State Assignment	
State	I_0	I_1	I_2	Y_1	Y_2	Y_1	Y_2
1	2	1	3	0	0	0	0
2	2	2	3	0	1	0	1
3	3	2	3	1	2	1	1
4	4	1	3	1	0	1	0
5	3	5	5	2	2	2	1
6	4	6	5	2	0	2	0
7	3	7	5	2	1	2	2

Fig. 12. Flow Table of M in Examples 4 and 5

$SP(\Pi_1)$		State	I_0	Input I_1	I_2	State Assignment Y_1
12	A	A	A	A	B	0
34	B	B	B	A	B	1
567	C	B	B	C	C	2

Fig. 13. Flow Table of M_1 is Examples 4 and 5.

		Input									Assign
Π_2	State	I_0A	I_0B	I_0C	I_1A	I_1B	I_1C	I_2A	I_2B	I_2C	Y_2
146	a	b	a	a	a	a	a	c	c	c	0
27	b	b	c	c	b	b	b	c	c	c	1
35	c	-	c	c	b	b	c	c	c	c	2

Fig. 14. Flow Table M_2 in Example 4.

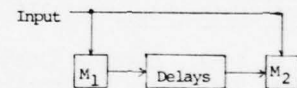


Fig. 15. Serial Connection with Insert Delay

		Input	
State	$I_k B_i$	$I_k B_j$	
s_i	s_j	-	
s_j	s_j	s_j	

Fig. 16. Entries of Flow Table M_2'

		Input									State Assignment
Π_2	State	I_0A	I_0B	I_0C	I_1A	I_1B	I_1C	I_2A	I_2B	I_2C	Y_2
1	2	-	-	-	1	1	-	3	-	-	-
2	2	-	-	-	2	2	-	3	-	-	-
3	-	3	3	-	-	2	-	3	3	-	-
4	-	4	4	-	1	-	-	-	-	-	-
5	-	-	3	-	-	-	5	-	-	5	-
6	-	-	4	-	-	-	6	-	-	5	-
7	-	-	3	-	-	-	7	-	-	5	-

Fig. 17. Flow Table of M_2' in Example 5

		Input									State Assignment
Π_2	State	I_0A	I_0B	I_0C	I_1A	I_1B	I_1C	I_2A	I_2B	I_2C	Y_2
146	a	b	a	a	a	a	a	b	-	b	0
235	b	b	b	b	b	b	b	b	b	b	1
7	c	-	-	b	-	-	c	-	-	b	2

Fig. 18. Flow Table of M_2 in Example 5

AD-A055 763

ILLINOIS INST OF TECH CHICAGO DEPT OF COMPUTER SCIENCE F/G 12/1
PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED L--ETC(U)
MAY 78 A S WOJCIK, R SWARTWOUT N00014-78-G-0019

UNCLASSIFIED

NL

3 OF 4
AD
A055763



Applications of Multivalued Threshold Logic in Large-Scale-Integrated, Digital Signal Processing Circuits

K. W. Current and D. A. Mow
Department of Electrical Engineering
University of California, Davis
Davis, California 95616

Modern large-scale-integrated (LSI) digital circuit design philosophy is approaching a technological crossroad. Improvements in photomasking and surface processing technologies have significantly improved the yield on very large ($>40,000 \text{ mil}^2$, $>5,000$ devices) chips. But even with these contributions, a majority of LSI chip designs are limited in their complexity by the number of metal signal lines that must be interconnected on the semiconductor surface, not by the number of devices. Will intensive efforts in device minimization and processing techniques provide significant returns in terms of increased circuit complexity? Or should alternatives in the coding and processing of digital electrical signals be explored? Much theoretical work has been done on multivalued threshold logic (e.g. [1-4]), and until recently few LSI circuit oriented realizations of these ideas have been reported [5-8]. Multivalued logic provides the attractive feature of an additional one, two or more logical values that each switching variable may assume. Standard binary signals assume only logical values ZERO and ONE, while four-valued logic may assume values ZERO, ONE TWO or THREE. Thus, use of four-valued logic may double the amount of information carried by each signal line. In a metallimitted design, this savings could be extremely important. Four-valued threshold logic circuits can provide a 50% reduction in the device count in some digital signal processing applications.

Four-valued threshold logic circuit realizations are most easily accomplished in either a current-steering logic similar to emitter-coupled logic (ECL) or with the current steering capabilities of integrated injection logic (I²L). Logical variables are developed as integer multiples of an easily duplicated reference amount of current. These currents are summed, differences, and duplicated to produce the desired logical results. In the ECL related technique, ECL compatible outputs are generated, while TTL compatible outputs can be generated in the I²L approach.

One important signal processing application for four-valued threshold logic is in digital counters and adders. A binary parallel counter is a multiple input circuit that counts the number of its inputs that are in a given state. This function can be performed with four-valued threshold logic full adder (QFA) circuits. A QFA accepts two base-four inputs A and B, and a binary carry input C_i and produces a two-quaternary-digit, base-four, output word CS, where C is the most significant digit. The table below summarizes implementations of binary parallel counters with binary full adders and QFA's. In general, 50% fewer intermediate signal lines (excludes binary inputs and outputs) and 50% fewer transistors and resistors are required.

Most applications of parallel counters are within the framework of another function; e.g. parallel

multipliers. A parallel counter is important in the development of an LSI digital output correlator, e.g. [9]. This correlator compared bit-by-bit two binary words and produces a binary count of the number of bits in agreement. Use of a QFA parallel counter in this application would reduce the total device count and chip area by about 14% and 18%, respectively.

References

1. Proc. of 1976 Internat. Symp. on Multivalued Logic, Logan, Utah, May 1976.
2. Proc. of 1975 Internat. Symp. on Multivalued Logic, Bloomington, Indiana, May 1975.
3. Proc. of 1974 Internat. Symp. on Multivalued Logic, Morgantown, W. Virginia, May 1974.
4. Proc. of 1973 Internat. Symp. on Multivalued Logic, Toronto, Canada, May 1973.
5. N. Friedman, C. Salama, P. Thompson, "Realization of a multivalued Integrated Injection Logic full adder," IEEE J. on Solid-State Circuits, pp. 532-534, Oct. 1977.
6. T. Dao, "Threshold I²L and its application to binary symmetric functions and multivalued logic," IEEE J. on Solid State Circuits, pp. 463-472, Oct. 1977.
7. K. C. Smith, "Circuits for multivalued logic," Proc. of 1976 Internat. Symp. on Multivalued Logic, pp. 30-43, May 1976.
8. J. K. Newton, "Implementation of Stefanelli multivalued parallel divider array," ibid, pp. 61-67.
9. R. Cheung, K. W. Current, E. E. Swartzlander, "A very high data rate digital correlation technique," 1976 Nat. Telecom. Conf. Rec., paper 52.4, Dallas, Texas, Nov. 1976.

		N (Inputs)					
		4	8	15	31	63	127
BINARY	FA's	4	11	26	57	120	
	Transistors and Resistors	144	396	936	2052	4320	
	Intermediate Variables	5	13	47	108	232	
FOUR-VALUED	FA's	1	3.5*	8	18.5*	39	
	Transistors and resistors, FA "A"	34	102	272	612	1326	
	Transistors and resistors, FA "B"	40	120	320	720	1560	
	Input conversion devices, 3N	21	45	93	189	381	
	OUTPUT conversions (devices)	2 (35)	2 (64)	3 (67)	3 (96)	4 (99)	
	Device total with FA "A"	90	234	432	920	1806	
	Device total with FA "B"	96	252	480	1028	2040	
	Intermediate variables	2	7	16	37	80	

*Half-adder produces four-valued count 0-3, 23 devices

Parts and line counts for N-Input Parallel Counters.

ITERATIVE REALIZATION OF MULTIVALUED LOGIC SYSTEMS

Vason P. Srinani

Department of Computer Science, Tennessee Technological University
Cookeville, Tennessee 38501

ABSTRACT

The realization of multivalued combinational functions and sequential machines by using arrays of one type of cells is considered. The algebra used for the multivalued logic system has two binary operations and a set of unary operations. Each of these operations is realized by a cellular array. The cells are combinational and implemented by using binary logic gates. The cells are also designed so that all unrestricted multiple faults in arrays of these cells are detectable. Multivalued combinational functions, storage elements and sequential machines are realized by interconnecting the arrays realizing the operations.

Index Terms

Multivalued logic, +gate, *gate, U gates, cell, cellular array, general fault, multiple faults, Fault detection.

I Motivation

The recent developments [1,2] in the processing technology of large scale integrated circuit (LSI) chips such as 16K and 64 K RAM chips, 64K and 256K CCD chips, and microprocessors have created a flurry of activities in realizing multivalued logic systems and microprocessors based on multivalued logic. Some of the major factors contributing to the above activities are the high circuit density per unit area of silicon, limitation on the number of pinouts, and the need for faster, powerful and economical computers. The success of RAM chips in memory systems have also shown that regularity in circuit structure and interconnection increases the yield of the fabrication process and also improves the diagnosability of the RAM chips [3]. It has also been observed that LSI and very large scale integrated circuits (VLSI) are almost impossible to test economically unless diagnostics is considered during the design phases of the chips [4].

The wide spread use of RAM chips, which consist of identical cells, has aroused interest in the design and fabrication of microprocessors based on cellular structures. If the individual

cells are designed to be diagnosable [5], then testing an array of these cells and consequently testing the entire microprocessor might be economically feasible.

An iterative realization of multivalued logic combinational functions and storage elements using cells of just one type, which are implemented by binary logic circuits based on existing and proven technologies such as TTL, I^2L , or ECL, is shown. The functionally complete multivalued algebra of Allen [6] is used for realizing the combinational functions and storage elements. Each operation of the algebra is realized by a cellular array. By suitably interconnecting the arrays realizing the operations the multivalued logic functions are realized.

If the number of logic levels, N , in the algebra is restricted to be a power of 2, then the N -valued clocked storage element of Sintonen [7] can be realized in a straightforward manner. This restriction also allows straightforward encoding to pinouts and decoding from pins.

Several algebras have been proposed for representing multivalued logic functions [8-13] and several realizations for combinational functions and N -valued storage elements using binary logic circuits [8-9] and non-binary logic circuits [10-12, 14] have been developed. Three of these approaches [9-11] are interesting. The vector Boolean algebra approach of Lee [9] is an extension of two valued Boolean algebra into a functionally complete multivalued algebra. The operations n -vector AND, n -vector OR, generalized complement, and rotation form a functionally complete set and each is realized by using binary logic circuits. Three canonical forms for representing multivalued combinational functions have been developed and the forms are realized by using gates corresponding to the operations. But the test generation for multiple faults is quite involved. The N -valued Boolean algebra of Dussault [10] with additional operations for functional completeness has two forms for representation. The realizations of the forms utilizes a decoder, binary circuits for realizing tables of subfunctions and an encoder to pinouts. The realizations are non-iterative, the encoder is quite complex, and there is no provision in the design for

diagnosability. Cellular realization of multivalued logic functions using multivalued q-functions is discussed by Kodandapani [11]. Three different types of cells are used in the realization and the cell design does not include features which might be helpful in diagnosing.

The algebra to be used in this work is described in Section II. The basic cell to be used for realizing the operations is shown in Section III. The realization of combinational functions, N-valued storage elements and sequential machines are described in Section IV. The testability of the cell is considered. Multiple fault detection in an array of these cells under general fault assumption is discussed in Section V.

II Algebraic System

The multivalued algebraic system considered in this discussion is the one developed by Allen [6]. Let $L = \{0, 1, 2, 3, \dots, (N-1)\}$ be the set of N logic values and k is the smallest integer such that $N \leq 2^k$. Each element, x, of L is represented by a k-tuple of zeros and ones such that the decimal value is x. Let + be a binary operator such that $x + y = \text{maximum}(x, y)$ and \cdot be a binary operator such that $x \cdot y$ (or xy) = minimum(x, y), where $x, y \in L$.

Let U be a set of unary operators ${}^a X^b$ such that

$${}^a X^b = \begin{cases} N-1 & \text{when } a \leq \text{logical value of } X \leq b \\ 0 & \text{when logical value of } X < a \text{ or} \\ & \text{logical value of } X > b, \end{cases}$$

$a, b \in L$ and $a \leq b$.

The system (L, +, \cdot , U) is a functionally complete multivalued algebra. The binary operations are associative and commutative.

III Realizing the Operations

The two binary operations and the set of unary operations are realized by using cellular arrays. The basic cell [15] is combinational and implemented by using binary logic circuits. The description of the cell is included for the sake of completeness.

The cell has three kinds of inputs. One input is from the external world, called primary input ($x_{i,j}$). The second input is from the left, called left pinout ($S_1 S_2$) and the third input is from the top, called top input ($y_{i,j}$). The cell has two outputs: one to the right, called right output ($P_1 P_2$) and the second to the bottom, called bottom output ($z_{i,j}$). All input and output lines are binary. We assume that the complement of $z_{i,j}$ ($\bar{z}_{i,j}$) is also available as bottom output. A typical cell along with the truth table realized by it is shown in Figure 1.

+ gate: This gate implements the + operation. It consists of a one-dimensional array of k cells with the interconnection pattern shown in Figure 2. If $x_i = (x_{i,1}, x_{i,2}, \dots, x_{i,k})$

and $y_i = (y_{i,1}, y_{i,2}, \dots, y_{i,k})$

are two multivalued logic variables, then $x_i + y_i$ can be computed by using y_i as the top input, x_i as the primary input and 00 as the left input to the array of k cells.

The array operates in the following manner. Starting from the leftmost cell, the values of $x_{i,j}$ and $y_{i,j}$, $1 \leq j \leq k$ are compared. If

$x_{i,j} = 1$ and $y_{i,j} = 0$ then

$x_i = (x_{i,1}, x_{i,2}, \dots, x_{i,k})$ is the maximum and the value of x_i is given as output ($z_{i,1}, z_{i,2}, \dots, z_{i,k}$). If $x_{i,j} = 0$ and $y_{i,j} = 1$ then $y_i = (y_{i,1}, y_{i,2}, \dots, y_{i,k})$ is the maximum and the value of y_i is given as output ($z_{i,1}, z_{i,2}, \dots, z_{i,k}$). If $x_{i,j} = 1$ (0) and $y_{i,j} = 1$ (0) then it is not possible to decide which is maximum from the first j cells in the array. The outputs $z_{i,k}$, $1 \leq k \leq j$ are 1s (0s). Subsequent cells are to be compared until either one of the two cases mentioned above happens or all cells are compared.

\cdot gate: This gate implements \cdot operation. It is similar to + gate except that the left input is 1 1.

Since the binary operations are associative, $(x_1 + x_2 + \dots + x_n)$ can be computed by a two-dimensional array of (n-1) cells and k columns with the left boundary set to a sequence of 0s and $(x_1 x_2 \dots x_n)$ can be realized by the above array if the left boundary is set to a sequence of 1s.

Ugates: Each elements, ${}^a X^b$, in the set U of unary operations is realized by a two-dimensional array of Figure 3. The left half of the first row compares the logic value of the variable X and a. If a is less than or equal to the logic value of X then the output of the right half of the row is (N-1). Otherwise the output is 0. The output of the left half of the row is the maximum of a and X and this is compared to b in the left half of the second row. If the logic value of X is less than or equal to b then the output of the right half of the second row is the output of the right half of the first row. Otherwise the output is 0. The output of the left half of the second row is ignored.

IV Realizing Logic Systems

The realization of combinational functions of multivalued logic is first discussed. Let x_1, x_2, \dots, x_n be the input variables that can take values from the set L and let F be the

output variable. Let α denote the n -tuple of logic values associated with input variables and the output be $F(\alpha) \in L$. Then, a table with N^n rows and $(n+1)$ columns can be used to specify the combinational function. The function can be represented by the form:

$$F = F(\alpha_1) \alpha_{1,1} \alpha_{1,1} \alpha_{1,2} \alpha_{1,2} \dots \\ \alpha_{1,n} \alpha_{1,n} + F(\alpha_2) \alpha_{2,1} \alpha_{2,1} \alpha_{2,2} \alpha_{2,2} \dots \\ \alpha_{2,n} \alpha_{2,n} + \dots + F(\alpha_i) \alpha_{i,1} \alpha_{i,1} \alpha_{i,2} \alpha_{i,2} \dots \\ \alpha_{i,n} \alpha_{i,n} + \dots + F(\alpha_{N^n}) \alpha_{N^n,1} \alpha_{N^n,1} \dots \\ \alpha_{N^n,2} \alpha_{N^n,2} \dots \alpha_{N^n,n} \alpha_{N^n,n}$$

where $\alpha_i = (\alpha_{i,1}, \alpha_{i,2}, \dots, \alpha_{i,n})$,

$$\alpha_{i,j} \in L, 1 \leq i \leq N^n, 1 \leq j \leq n,$$

such that the N -ary value of the n -tuple is $(i-1)$. The number of terms in the above form can be reduced by using the minimization algorithm of Allen [6].

The above form can be realized by using $+$ gates, \cdot gates, U gates, and the logical constants of L .

A combinational function for a 3 valued system is shown in Table 1 [6].

Table 1

x_1	x_2	F
0	0	2
0	1	2
0	2	0
1	0	1
1	1	2
1	2	2
2	0	0
2	1	2
2	2	2

$$F = 2 \cdot 0_{x_1}^0 \cdot 0_{x_2}^0 + 2 \cdot 0_{x_1}^0 \cdot 1_{x_2}^1 + 1 \cdot 1_{x_1}^1 \cdot 0_{x_2}^0 + \\ 2 \cdot 1_{x_1}^1 \cdot 1_{x_2}^1 + 2 \cdot 1_{x_1}^1 \cdot 2_{x_2}^2 + 2 \cdot 2_{x_1}^2 \cdot 1_{x_2}^1 + \\ 2 \cdot 2_{x_1}^2 \cdot 2_{x_2}^2.$$

After minimizing,

$$F = 2 \cdot 0_{x_1}^0 \cdot 0_{x_2}^1 + 1 \cdot 1_{x_1}^1 + 2 \cdot 1_{x_1}^2 \cdot 1_{x_2}^2.$$

A realization of the above form is shown in Figure 4.

Realizing multivalued logic sequential machines is now considered. This requires the availability of an N -state storage element in addition to the combinational circuits for realizing next state and output functions. Let the complement of a multivalued logic variable X be denoted by \bar{X} whose value is equal to $(N-1)$ minus the logic value of X . Let $N=2^k$. Then, the clocked N -state storage element of Sintonen [7] can be realized by using two \cdot gates with two inputs, one $(N-1)_{x(N-1)}$ gate, one $(N-1)_{x(N-1)}$ gate and one $+$ gate with two inputs. The realization of $(N-1)_{x(N-1)}$ is similar to the realization of the gate $(N-1)_{x(N-1)}$ which is in the set of U gates, except that the output is from the complement side of $z_{i,j}$ (i.e. $\bar{z}_{i,j}$). The interconnection of gates realizing the N -valued storage element is shown in Figure 5.

The next state function of the storage element is

$$Q(t+1) = (N-1)_{x_1} (N-1)_{x_2} + \overline{(N-1)_{x_1} (N-1)} Q(t), \\ \text{where } x_1, x_2 \in L.$$

The clocking state is $(N-1)$ and x_1 is the clocking input.

The realization of sequential machines is illustrated by a synchronous mod N^2 counter, where $N = 4$. The state table is shown in [7] and excitation functions are:

$$Q_1: \quad x_1 = 3 \\ x_2 = 1 \cdot 0_{Q_1}^0 + 2 \cdot 1_{Q_1}^1 + 3 \cdot 2_{Q_1}^2 \\ Q_2: \quad x_1 = Q_1 \\ x_2 = 1 \cdot 0_{Q_2}^0 + 2 \cdot 1_{Q_2}^1 + 3 \cdot 2_{Q_2}^2.$$

The block diagram for realizing the synchronous mod N^2 counter is shown in Figure 6.

V Fault Detection

A cell is considered faulty if the behavior of the cell changes to any combination function other than the one shown in Figure 1. This assumption is known as general fault assumption [16, 17]. The detection of faults in one and two-dimensional arrays when one or more cells have failed is considered.

The following conditions are necessary and sufficient for an array to be testable [16, 17]:

- It should be possible to apply to the input terminals of any cell, a complete set of tests for detecting all faults in the cell, independent of the position of the cell in the array.
- For each such test, it should be possible to propagate the effect of the fault to an observable output.

Based on the above conditions Prasad [5] had developed a simple design criterion for multiple fault detection under general fault assumption. Let a cell be characterized by a function $g: X \times Y \rightarrow X$, where X is the set of left inputs and Y is the set of top inputs. Let there exist an element "b" belonging to Y such that $g(s,b) = \mu(s)$ for some permutation μ on X , where $s \in X$. Then the test for one-dimensional array is [5],

$$T^k = \{(s, y_{1,1}, y_{1,2}, \dots, y_{1,j}, \dots, y_{1,k}) \mid$$

$$s \in X, y_{1,1} = y_{1,2} = \dots = y_{1,(j-1)} = y_{1,(j+1)}$$

$$= \dots = y_{1,k} = b, y_{1,j} \in Y \text{ for some } j\}.$$

We note from Figure 1 that for $b = 0$, $g(s,0) = (s,0)$ when $x_{i,j} = 0$, and for $b = 1$, $g(s,1) = (s,1)$ when $x_{i,j} = 1$, with μ the identity permutations on X .

Therefore one-dimensional array of cells shown in Figure 2 is testable.

The test for $x_{i,j} = 0$ when $k = 2$ is

$$T^2 = \begin{matrix} s \\ y_{1,1} \\ y_{1,2} \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 3 & 3 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

and the test for $x_{i,j} = 1$ is

$$T^2 = \begin{matrix} s \\ y_{1,1} \\ y_{1,2} \end{matrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 3 & 3 & 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Let there exist $a \in X$, $b \in Y$ such that $g(s,b) = (\mu(s),b)$ and $g(a,c) = (a,\nu(c))$, where $c \in Y$ and for some arbitrary permutations ν on X and μ on Y . Then the test for a two-dimensional array is [5].

$$TP,k = \{(s_{1,1}, s_{2,1}, \dots, s_{i,1}, \dots, s_{p,1},$$

$$y_{1,1}, y_{1,2}, \dots, y_{1,j}, \dots, y_{1,k}) \mid$$

$$\text{for some } i \text{ and } j, s_{q,1} = a, q \neq i, y_{1,m}$$

$$= b, m \neq j, s_{i,1} \in X, y_{1,j} \in Y\}$$

We note from Figure 1 that for $a = 2$, $b = 0$, $g(s,0) = (s,0)$ and $g(2,c) = (2,c)$ when $x_{i,j} = 0$, and for $a = 2$, $b = 1$, $g(s,1) = (s,1)$ and $g(2,c) = (2,c)$ when $x_{i,j} = 1$ with μ the identity permutation on X and ν the identity permutation on Y . Therefore two-dimensional arrays of cells are testable. The test set when $p = 2$ and $k = 2$ is shown below.

VI Remarks

Realizing multivalued logic systems by using gates constructed from arrays of cells has been discussed. A cellular array can be tested for multiple unrestricted faults by a test set consisting of $2[8pk - 6(pk - p - k) - 4k - 2p + (p-1)(k-1)]$ elements, calculated by using [5], where p is the number of rows and k the number of columns in the array. This non-exhaustive testing of arrays of cells reduces the test time for LSI chips using lasers [18] at the production stage and thus removing one of the major obstructions to the production of low cost LSI.

The test set for $x_{i,j} = 0$ when $p = 2$ and $k = 2$ is,

$$T^{2,2} = \begin{matrix} s_{1,1} \\ s_{2,1} \\ y_{1,1} \\ y_{1,2} \end{matrix} \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 0 & 0 & 1 & 1 & 3 & 3 & 0 & 1 & 3 \\ 0 & 0 & 1 & 1 & 2 & 2 & 3 & 3 & 0 & 1 & 2 & 3 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

and the test set for $x_{i,j} = 1$ is,

$$T^{2,2} = \begin{matrix} s_{1,1} \\ s_{2,1} \\ y_{1,1} \\ y_{1,2} \end{matrix} \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 0 & 0 & 1 & 1 & 3 & 3 & 0 & 1 & 3 \\ 0 & 0 & 1 & 1 & 2 & 2 & 3 & 3 & 0 & 1 & 2 & 3 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

References

1. T.H.P. Chang, et al, "Electron-beam lithography draws a finer line", Electronics, May 12, 1977, Vol. 50, No. 10, pp. 89-98.
2. L. Altman and C. L. Cohen, "Japan Presses innovations to reach VLSI goal", Electronics, June 9, 1977, Vol. 50, NO. 12, pp. 99-122.
3. V. P. Srimi, "Fault location in a semiconductor RAM unit", to appear in IEEE Transactions on Computers, April 1978.
4. R. Spillman, "Single stuck type fault detection in multi-valued combinational circuits", Proc. of 1976 Symposium on Multiplevalued logic design, pp. 97-101.
5. B. A. Prasad and F. G. Gray, "Multiple fault detection in arrays of combinational cells", IEEE Transactions on Computers, Vol. c-24, No. 8, Aug. 1975, pp. 794-802.
6. C. M. Allen and D. D. Givone, "A minimization technique for multiple-valued logic systems", IEEE Transactions on Computers, Vol. c-17, No. 2, Feb. 1968, pp. 182-184.
7. L. Sintonen, "A clocked multivalued flip-flop", IEEE Transactions on Computers, Vol. c-26, No. 3, March 1977, pp. 292-294.
8. B. Benjauthrit and I. S. Reed, "Galois switching functions and their applications", IEEE Transactions on Computers, Vol. c-25, No. 1 Jan. 1976, pp. 78-86.
9. S. C. Lee, "Vector Boolean algebra and calculus", IEEE Transactions on Computers, Vol. c-25, No. 9, Sept. 1976, pp. 865-874.
10. J. Dussault, G. Metze, and M. Krieger, "A multivalued switching algebra with Boolean properties", Proc. of 1976 Symposium on Multiplevalued logic design, pp. 68-73.
11. K. L. Kodandapani and R. V. Setlur, "A cellular array for multivalued logic functions", Proc. of 1974 Symposium on Multiplevalued logic design, pp. 529-544.
12. R. C. Braddock, G. Epstein, and H. Yamanaka, "Multiplevalued logic design and applications in binary computers", Proc. of 1971 Symposium on Multivalued logic design, Buffalo NY, pp. 13-25.
13. Z. G. Vranesic, E. S. Lee, and K. C. Smith, "A manyvalued algebra for switching systems", IEEE Transaction on Computers, Vol. c-19, No. 10, Oct. 1970, pp. 964-971.
14. D. M. Miller and J. C. Muzio, "A ternary cellular array", Proc. of 1974 Symposium on Multivalued logic design, pp. 469-482.
15. V. P. Srimi, "Realization of fuzzy forms", IEEE Transaction on Computers, Vol. c-24, No. 9, Sept. 1975, pp. 941-943.
16. A. D. Friedman and P. R. Menon, Fault detection in digital circuits, Prentice-Hall Inc., Englewood Cliffs, NJ 1971.
17. W. H. Kautz, "Testing for faults in cellular logic arrays," Proc. 8-th annual Symp. on Switching and automata theory, 1967, pp. 161-174.
18. J. G. Smith and H. E. Oldham, "Laser testing of integrated circuits", IEEE Journal of Solid State Circuits, Vol. sc-12, No. 3, June 1977, pp. 247-252.

Acknowledgement

The author is thankful to Sue for her encouragement and sustenance.

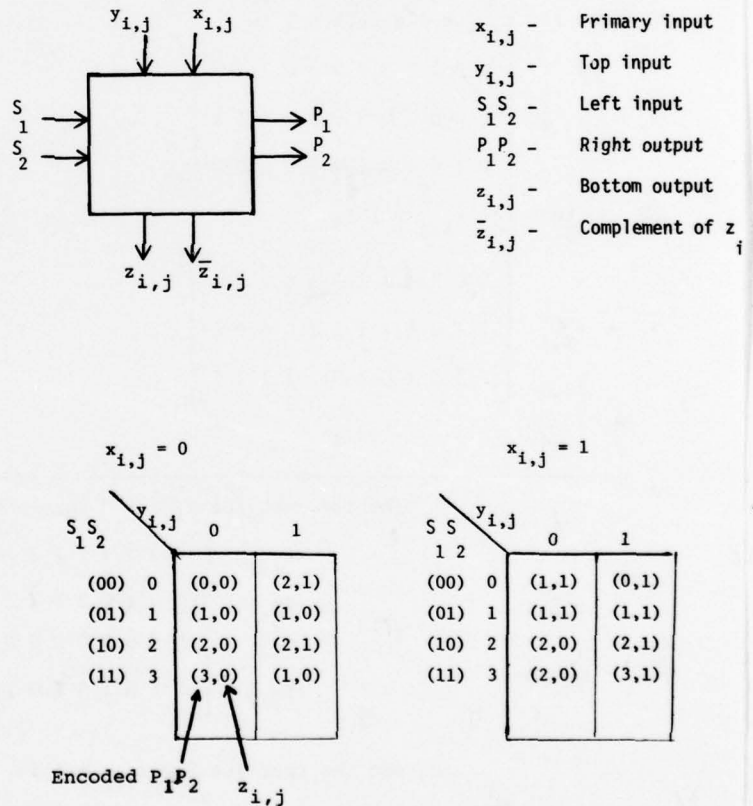
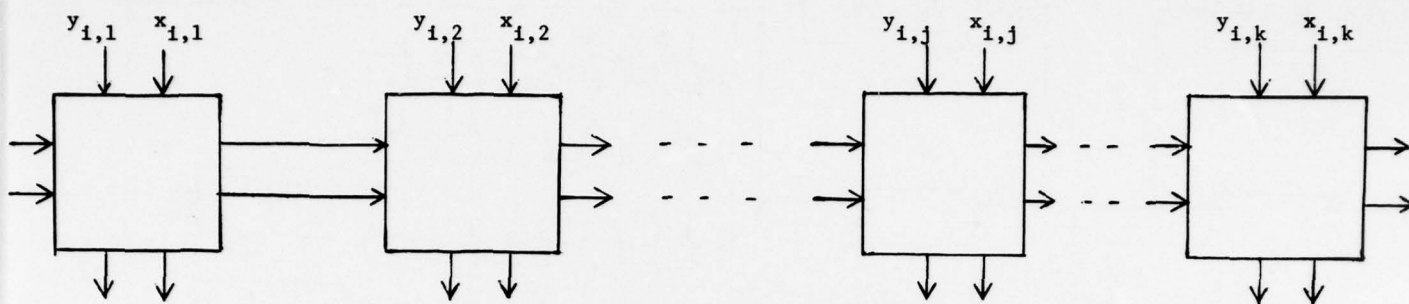


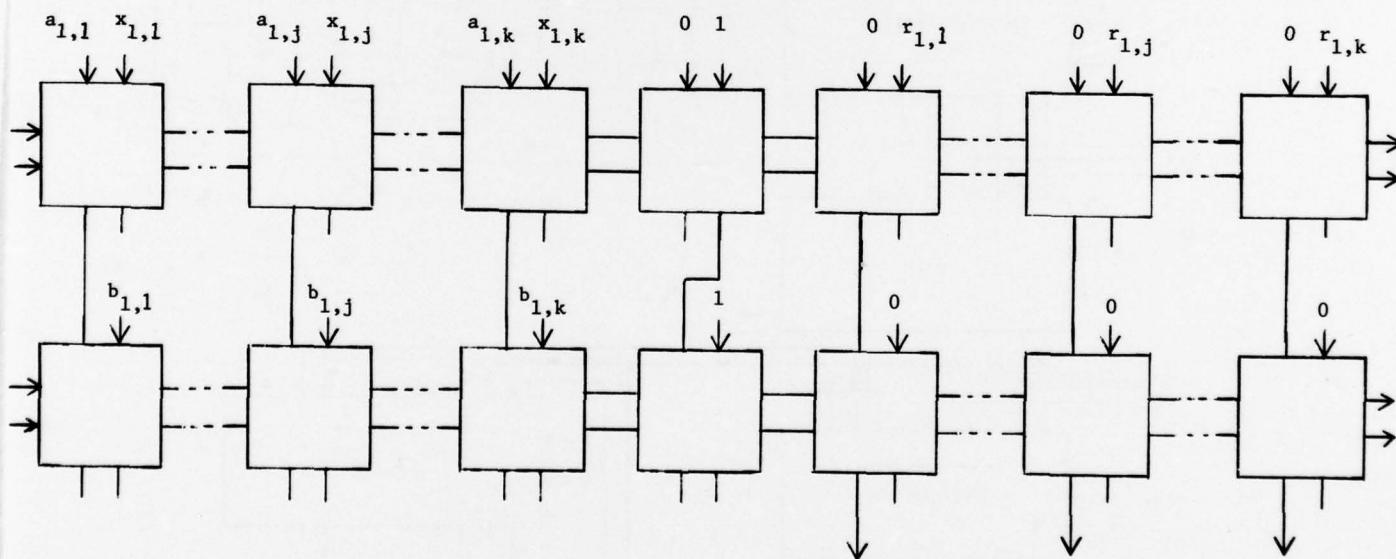
Figure 1. A typical cell and its truth table



$$x_i = (x_{i,1}, x_{i,2}, \dots, x_{i,j}, \dots, x_{i,k})$$

$$y_i = (y_{i,1}, y_{i,2}, \dots, y_{i,j}, \dots, y_{i,k})$$

Figure 2. Realizing + operation



$$a = (a_{1,1}, a_{1,2}, \dots, a_{r,j}, \dots, a_{1,k})$$

$$b = (b_{1,1}, b_{1,2}, \dots, b_{1,j}, \dots, b_{1,k})$$

$$N-1 = (r_{1,1}, r_{1,2}, \dots, r_{1,j}, \dots, r_{1,k})$$

$$x = (x_{1,1}, x_{1,2}, \dots, x_{1,j}, \dots, x_{1,k})$$

Figure 3. Realizing a_x^b

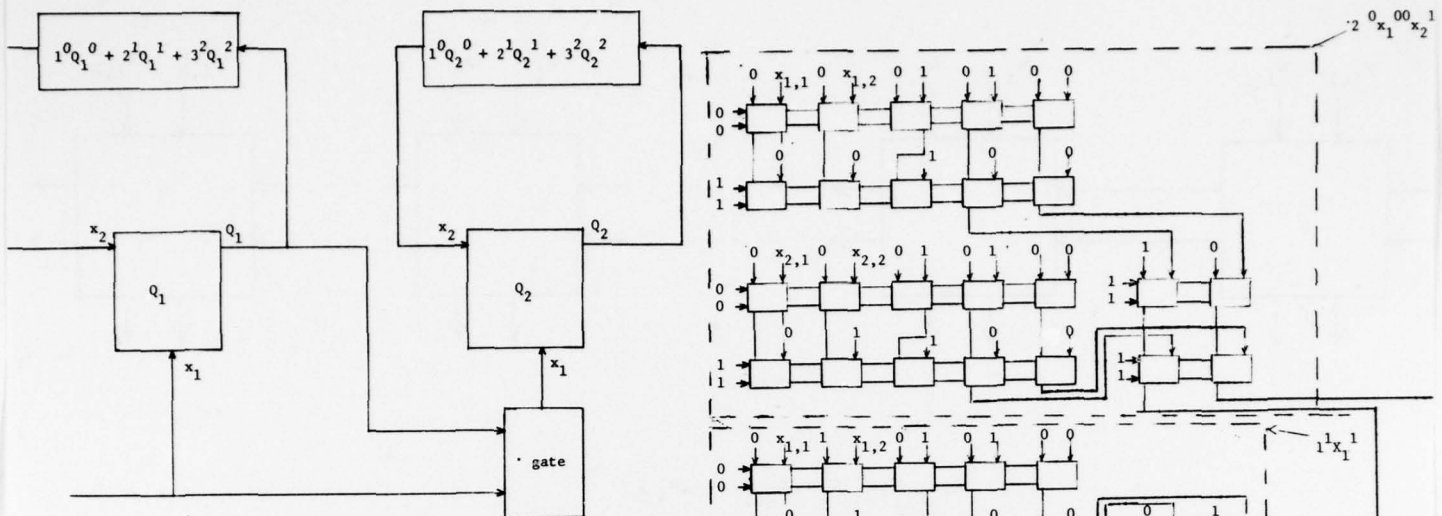


Figure 6. Block diagram for synchronous mod N^2 counter

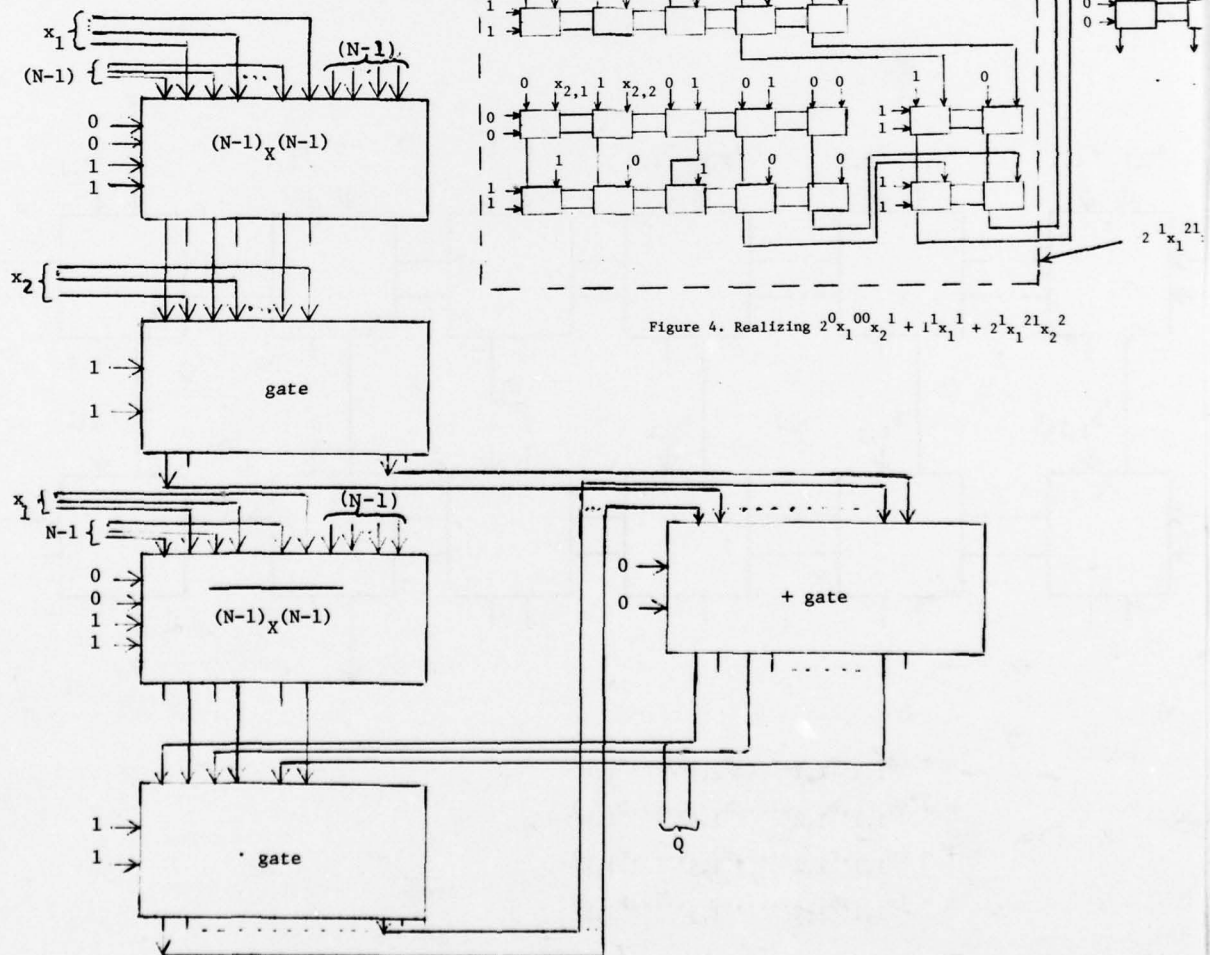


Figure 4. Realizing $2^0 x_1^{00} x_2^1 + 1^1 x_1^1 + 2^1 x_1^{21} x_2^2$

Figure 5. Realizing N-valued storage element

DETERMINATION OF THE FITTEST NUMBER OF TRUTH-VALUES AND CANONICAL FORMS OF LOGICAL FUNCTIONS FOR A MANY-VALUED AXIOM SET BY A COMPUTER

Motinori Goto and Shinji Kao and Tomoko Ninomiya

Meiji University
Tokyo, Japan

Usually the number of truth-values and the meanings of operators in a given axiom set for many-valued logic are not defined explicitly but done implicitly by many formulas derived from the set. However, it is very laborious to determine them. In this paper, axioms and undefined operators are treated as logical equations and unknown logical functions respectively. General solutions of those equations give general truth tables for undefined operators. For example, M. Wajsberg's axiom set is solved by a computer. The fittest number of truth-values are determined in relation to the construction of the canonical form of logical functions.

1 Introduction

Usually the number of truth-values and the meanings of operators in a given axiom set for mathematical logic are not defined explicitly but done implicitly by many formulas derived from the set. However, especially for a many-valued axiom set, it is very laborious to derive the truth tables for those undefined operators and their fittest number of truth-values.

In the preceding paper⁵, undefined operators were treated as unknown logical functions and axioms were treated as logical equations. General solutions^{3,4} of those equations led us to the general truth tables for undefined operators by a computer and independence and consistency of the axiom set were proved as the byproducts of the derivation of the general solution.

In this paper, as effective applications of the above-mentioned method, canonical forms of logical functions which satisfy the general solutions of the truth tables for a given axiom set are derived, where the fittest number of truth-values are also determined in relation to the construction of the canonical forms.

Since an axiom set for many-valued logic has the general solutions of the truth tables which consist of several subsets for the undefined operators, if we add any other independent axiom, some subsets of solutions may be cancelled. Then the general solutions are completely cancelled only by adding an axiom which has no subset in common with the original general solutions.⁵ Such results mean that the definition of the strong completeness in the classical two-valued logic is inadequate to many-valued logic.

It is necessary for explanation to repeat the first part of the preceding paper,⁵ but the FORTRAN program to solve the logical equations is omitted in this paper.

2 General Construction of Axioms

Axioms are assumed to have the form $A \supset B$. The logical expressions A and B consist of any variables X, Y, Z, \dots of a finite number and their connectives (\quad) , \supset and \cup or \cap , of a finite number, where (\quad) are parenthesis and \supset and \cup are two-term operators, while \cap is a single-term operator. Of course, if necessary, we can add other undefined operators and other auxiliary operators are to be defined in terms of those undefined ones as the following examples: $A \cup B \equiv (A \supset B) \supset B, A \cap B \equiv \cap (\cap A \cup \cap B)$, (2.1) where \cup and \cap are "Three-valued disjunction and conjunction", \supset and \cap being "Three-valued implication and negation".

3 Truth-Value

A, B, X, Y, Z, \dots take only one of the truth-values $1, 2, \dots, N_0$, where 1 is designated as "Truth".

" X takes the truth-value x " is represented by $X=x$ (3.1)

and simply by the two-valued expression X^x . Then $X^x = (X = x)$. (3.2)

4 Connectives for Classical Two-Valued Logic

Negation \bar{A} or $\sim A$. Disjunction $A \vee B$.

Conjunction $A \cdot B$. Implication $A \rightarrow B$.

Equivalence $A \leftrightarrow B$.

Identical equivalence $A \equiv B$,

which means " A and B are equivalent for every truth-value of the constituents X, Y, Z, \dots ".

5 Representation of the Undefined Operators

Using the form (3.2), N_0 -valued implication with the truth-value " l " is expressed as the following two-valued expression

$$(A \supset B)^l \equiv \text{IMP } (A, B)^l \equiv \bigvee_{m=1}^{N_0} \bigvee_{n=1}^{N_0} I_{mn}^l \cdot A^m \cdot B^n. \quad (5.1)$$

Similarly for N_0 -valued disjunction,

$$(A \cup B)^l \equiv \text{DISJ } (A, B)^l \equiv \bigvee_{m=1}^{N_0} \bigvee_{n=1}^{N_0} D_{mn}^l \cdot A^m \cdot B^n. \quad (5.2)$$

For N_0 -valued negation,

$$(7A)^{\ell} \equiv \text{NEG } (A)^{\ell} \equiv \bigvee_{m=1}^N N_m \cdot A^m. \quad (5.3)$$

6 Example of Axiom Set for Three-Valued Logic

In this paper, for a typical example, the three-valued axiom set is treated. Similarly we can treat a two-valued axiom set and a four-valued one.

$$A^1 \cdot (A \supset B)^1 \rightarrow B^1 \equiv 1, \quad (6.0)$$

$$[X \supset (Y \supset X)]^1 \equiv 1, \quad (6.1)$$

$$[(X \supset Y) \supset ((Y \supset Z) \supset (X \supset Z))]^1 \equiv 1, \quad (6.2)$$

$$[(7Y \supset 7X) \supset (X \supset Y)]^1 \equiv 1, \quad (6.3)$$

$$[(X \supset 7X) \supset X]^1 \equiv 1, \quad (6.4)$$

where Eqs. (6.1)~(6.4) represent two-valued expressions of M. Wajsberg's axioms ^{1,2} for three-valued logic and Eq. (6.0) corresponds to "modus ponens", which means "If the truth-values of A and A \supset B are 1, then that of B is 1".

7 Transformation of the Axioms by IMP and NEG

Eqs. (6.0)~(6.4) are transformed to Eqs. (7.0)~(7.4) by means of IMP and NEG in Eqs. (5.1) and (5.3).

$$\overline{A^1} \vee \text{IMP } (A, B)^1 \vee B^1 \equiv 1. \quad (7.0)$$

$$\text{IMP } (X, \text{IMP } (Y, X))^1 \equiv 1, \quad (7.1)$$

$$\text{IMP } (\text{IMP } (X, Y), \text{IMP } (\text{IMP } (Y, Z), \text{IMP } (X, Z)))^1 \equiv 1, \quad (7.2)$$

$$\text{IMP } (\text{IMP } (\text{NEG } (Y), \text{NEG } (X)), \text{IMP } (X, Y))^1 \equiv 1, \quad (7.3)$$

$$\text{IMP } (\text{IMP } (\text{IMP } (X, \text{NEG } (X)), X), X)^1 \equiv 1. \quad (7.4)$$

8 Expression of the Axioms in terms of the Coefficients I_{mn}^{ℓ} and N_m^{ℓ}

Replacing IMP and NEG in Eqs. (7.0)~(7.4) by I_{mn}^{ℓ} and N_m^{ℓ} in Eqs. (5.1)~(5.3), the following Eqs. (8.0)~(8.4) are easily obtained for any truth-values X, Y, Z.

$$\overline{I_{12}^1} \vee I_{13}^1 \therefore I_{12}^1 = I_{13}^1 = 0 \quad (8.0)$$

$$\bigwedge_{x,y=1}^3 \bigvee_{j=1}^3 I_{xy}^1 \cdot I_{yx}^j = 1, \quad (8.1)$$

$$\bigwedge_{x,y,z=1}^3 \bigvee_{j,k,\ell,m=1}^3 I_{jk}^1 \cdot I_{xy}^j \cdot I_{\ell m}^k \cdot I_{yz}^{\ell} \cdot I_{xz}^m = 1, \quad (8.2)$$

$$\bigwedge_{x,y=1}^3 \bigvee_{j,k,\ell,m=1}^3 I_{jk}^1 \cdot I_{\ell m}^j \cdot N_y^{\ell} \cdot N_x^m \cdot I_{xy}^k = 1, \quad (8.3)$$

$$\bigwedge_{x=1}^3 \bigvee_{j,k,\ell=1}^3 I_{jx}^1 \cdot I_{kx}^j \cdot I_{x\ell}^k \cdot N_x^{\ell} = 1, \quad (8.4)$$

where x, y, z mean the truth-values of independent variables x, y, z and j, k, ℓ , m mean those of terms IMP and NEG. As shown in Eqs. (8.1)~(8.4), the order of the positions of the factors I_{mn}^{ℓ} and N_m^{ℓ} is always the same as that of the corresponding IMP (M, N) ^{ℓ} and NEG (M) ^{ℓ} . Then Eqs. (7.1)~(7.4) are formally transformed to Eqs. (8.1)~(8.4) without any calculation. Eqs. (8.0)~(8.4) have 24 unknown variables which satisfy 3⁸ conditions.

9 Solutions of Eqs. (8.0)~(8.4) by a Computer

Simultaneous logical Eqs. (8.0)~(8.4) are easily solved using FORTRAN by a computer. In FORTRAN, in place of "x, y, z, ..., " we use corresponding capital letters. For example, I_{yx}^j and N_y^{ℓ} may be expressed in FORTRAN as follows:

$$I(P, IBC), \quad (9.1)$$

$$IBC = 9 * J + 3 * Y + X \quad (9.2)$$

$$N(P, NA), \quad (9.3)$$

$$NA = 9 * L + 3 * Y. \quad (9.4)$$

For any one set of X, Y, Z in a certain axiom, P means the ordinal number of any one of the subsets of the solutions of I_{mn}^{ℓ} , N_m^{ℓ} for ($\ell = 1, 2$; $m = 1, 2, 3$; $n = 1, 2, 3$) such as $P = 1, 2, \dots$, R, where I_{mn}^3 , N_m^3 are obtained by $I_{mn}^3 = I_{mn}^1 \cdot I_{mn}^2$,

$$N_m^3 = N_m^1 \cdot N_m^2.$$

According to Eq. (8.0), the following initial values should be given.

$$I(P, 14) = I(P, 15) = 0, \quad (9.5)$$

$$I(P, IBC) = 2 \text{ for } IBC \neq 14, 15, \quad (9.6)$$

$$N(P, NA) = 2, \quad (9.7)$$

where "14" and "15" in Eq. (9.5) mean $9 * 1 + 3 * 1 + 2 = 14$, $9 * 1 + 3 * 1 + 3 = 15$, (9.8)

according to Eq. (8.0), and "2" in Eqs. (9.6)~(9.7) means "Indeterminate".

At first, Eq. (8.1) is to be solved as shown in Table 9.1, for XYZJKLM...B = 1101000...0, where "11010...0" means X=Y=J=1 and 0's mean "Z, K, L, M, ..., B are nothing", and "10022222 222222" means that

$$I_{11}^1 = 1, I_{11}^2 = 0; I_{12}^1 = I_{13}^1 = 0, \quad (9.9)$$

other I's and N's being 2.

Let us call such solutions "Particular solutions" which are expressed as I1 (P1, IBC) and N1 (P1, NA), where P1 indicates: the ordinal number of the subset of particular solutions.

The solution $I_{11}^1 = 1$ is derived from $I_{xj}^1 = 1$ for $x=j=1$ i.e. $x=j=1$. Next, $I_{yx}^j = 1$ for $X=Y=J=1$ gives the same solution which yields no contradiction to I_{xj}^1 .

For other truth-values of J,K,L,M there are contradictions among

$$I_{12}^1 = I_{13}^1 = 0, I_{xy}^1 = 1, I_{yx}^1 = 1.$$

In any subset of particular solutions, if any contradiction would be found between a preceding solution and a succeeding one, then that subset should be cancelled automatically by a computer.

In this example, the first set of provisional solutions I(P, IBC) and N(P, NA) is the same one as I1 and N1.

Next for XYZJK..B = 12010...0, $I_{21}^1 = 1$ and then $I_{21}^2 = 0$.

These new particular solutions are different from $I_{21}^1 = 2$ and $I_{21}^2 = 2$ in the first set of the provisional solution I and N, but not contradictory. Then the combination of the first set of the provisional solutions and this new subset of particular solutions lead us to the set of the provisional solutions which is the same as the new particular solutions.

For the above-mentioned combinations, the following special four-valued logic may be used.

Truth-value: $\emptyset, 0, 1, 2$.

" \emptyset " means "Contradiction".

"0" and "1" mean "Truth-values in the classical two-valued logic".

"2" means "Indeterminate".

Conjunction: $\emptyset \cdot \emptyset = \emptyset \cdot 0 = \emptyset \cdot 1 = \emptyset \cdot 2 = \emptyset$,
 $0 \cdot 0 = 0, 0 \cdot 1 = \emptyset, 0 \cdot 2 = 0$,
 $1 \cdot 1 = 1 \cdot 2 = 1, 2 \cdot 2 = 2$ (9.10)

Disjunction: $\emptyset \vee \emptyset = \emptyset, \emptyset \vee 0 = 0, \emptyset \vee 1 = 1, \emptyset \vee 2 = 2$,
 $0 \vee 0 = 0, 0 \vee 1 = 1, 0 \vee 2 = 2$,
 $1 \vee 1 = 1, 1 \vee 2 = 2, 2 \vee 2 = 2$. (9.11)

For the above-mentioned four-valued operations, the following 2-bit expressions may be used.

$\emptyset = 00, 0 = 01, 1 = 10, 2 = 11$. (9.12)

We can easily derive (9.10) by the bit-wise conjunction of (9.12) and also (9.11) by the bit-wise disjunction of (9.12).

Similarly for XY=21, multiplying the preceding provisional solution with P=1 by the particular ones with P1 = 1, 2, we can obtain the new provisional ones with P = 1,2.

During these four-valued multiplications, if " \emptyset " is found in some subset of provisional ones, then that subset should be cancelled and its ordinal number "P" should be shifted to the next subset.

As shown in Table 9.1, in the set of the provisional solutions with P = 1, 2 for XY = 33,

$$I_{11}^1 = I_{21}^1 = I_{31}^1 = 1, I_{12}^1 = I_{13}^1 = I_{11}^2 = I_{21}^2 = I_{22}^2 = I_{31}^2 = I_{32}^2 = N_1^1 = 0.$$

These results indicate truth-values of I_{mn}^e and N_m^e which have been settled to "0" or "1" up to now. Let us indicate such settled results as I0 (IBC) and N0 (NA).

I0 (IBC) and N0 (NA) may be derived by the following four-valued disjunction \vee according to Eq. (9.11):

$$I0(IBC) = \bigvee_{p=1}^R I(P, IBC), N0(NA) = \bigvee_{p=1}^R N(P, NA), \quad (9.14)$$

where R is the last ordinal number of the set of the provisional solutions.

Then the next I1 (P1, IBC) and N1(P1, NA) should be multiplied by these preceding I0(IBC) and N0(NA) according to Eq.(9.10) and if " \emptyset " is found then such subset of particular solutions I1 and N1 should be cancelled.

The general solutions of I and N are shown in Table 9.1, which is the last set of provisional solutions of I and N for X=3 in the last axiom, consisting of the ten subsets.

These general solutions are transformed to the truth-values of IMP(A,B) and NEG(A) for all combinations of A and B as shown in Table 9.1, according to the following formulas.

$$\begin{aligned} \text{IMP}(A,B) &= 3 - 2 \cdot I(P, IAB) - I(P, IIAB), \\ \text{NEG}(A) &= 3 - 2 \cdot N(P, NA) - N(P, NNA), \end{aligned} \quad (9.15)$$

where

$$\begin{aligned} IAB &= 9 + 3 \cdot A + B, NA = 9 + 3 \cdot A, \\ IIAB &= 9 \cdot 2 + 3 \cdot A + B, NNA = 9 \cdot 2 + 3 \cdot A. \end{aligned}$$

10. Construction of a Canonical Form

Table 10.1

P	X	7X1	7X2	7X3
5	1	1	3	3
	2	3	1	3
	3	3	3	1
4	1	1	3	3
	2	3	3	1
	3	3	3	1

Table 10.1 shows the truth-values of the following three basic functions

$$\begin{aligned} X1 &\equiv X \supset X, X3 \equiv 7X \supset X, \\ X2 &\equiv X3 \supset 7X1, \end{aligned} \quad (10.1)$$

for X=1, 2, 3 of the subset P = 5, 4 of the general solution in Table 9.1.

For P = 5,

$$\begin{aligned} 7Xn &= 1 \text{ for } X = n \ (n = 1, 2, 3) \\ &= 3 \text{ for } X \neq n. \end{aligned} \quad (10.2)$$

Then, if we use the following function,

$$\begin{aligned} F13 &\equiv (C3 \supset X3) \supset (C1 \supset X1), \\ F2 &\equiv 7X2 \cap C2 \equiv 7(X2 \cup 7C2) \\ &\equiv 7((X2 \supset 7C2) \supset 7C2), \end{aligned} \quad (10.3)$$

where C1, C2, C3 may be designated logical constants or any other logical functions, we have

$$\begin{aligned} F13 &= C1 \text{ for } X = 1, \\ &= C3 \text{ for } X = 3, \\ &= 3 \text{ for } X = 2; \\ F2 &= C2 \text{ for } X = 2, \\ &= 3 \text{ for } X \neq 2, \end{aligned} \quad (10.4)$$

$$\begin{aligned} F123 &\equiv F13 \cup F2 \\ &\equiv (F2 \supset F13) \supset F13 \\ &= Cn \text{ for } X = n. \end{aligned} \quad \begin{aligned} (10.5) \\ (10.5') \end{aligned}$$

Eq. (10.5') means that this expression F123 has the property of a canonical form of any logical function F(X)

which satisfy

$$\begin{aligned} F(n) &= Cn \\ \text{for } P &= 5. \end{aligned} \quad (10.6)$$

Table 9.1

	I(1FC)	I(2BC)	N(1A)	N(2A)	
A			123	123	
B	111222333	111222333			
C	123123123	123123123			
XVZJKLMHGFEDCH,					P1, P
11010000000000,	100222222,022222222,	222 222,	1		1
	100222222,022222222,	222 222,		1	
12010000000000,	100122222,022022222,	222 222,	1		1
	100122222,022022222,	222 222,		1	
13010000000000,	100222122,022222022,	222 222,	1		1
	100122122,022022022,	222 222,		1	
21020000000000,	200212222,212202222,	222 222,	1		
21030000000000,	200221222,202220222,	222 222,	2		
	100112122,012002022,	222 222,		1	
	100121122,002020022,	222 222,		2	
22010000000000,	200112222,222002222,	222 222,	1		
22030000000000,	200201222,222200222,	222 222,	2		
	100112122,012002022,	222 222,		1	
	100111122,002000022,	222 222,		2	
	100101122,002000022,	222 222,		3	
omitted					
20021100000000,	200112222,212002222,	212 202,	1		
20021200000000,	200212222,212202222,	202 212,	2		
20021300000000,	200211222,212200222,	202 202,	3		
20031100000000,	200122212,202022202,	212 202,	4		
20031200000000,	200212212,202202202,	202 212,	5		
20031300000000,	200221212,202220202,	202 202,	6		
20032300000000,	200200212,222201202,	202 202,	7		
	100111111,011000000,	011 100,		1	
	100111111,011000000,	011 000,		2	
	100110111,011001000,	001 010,		3	
	100111111,010000000,	011 100,		4	
	100111111,010000000,	011 000,		5	
	100110111,010001000,	001 010,		6	
	100111101,010000000,	010 100,		7	
	100111111,001000000,	011 100,		8	
	100111111,001000000,	011 000,		9	
	100111111,000000000,	011 100,		10	
	100111111,000000000,	011 000,		11	
30021100000000,	200221122,221220022,	221 220,	1		
30021300000000,	200221221,221220220,	220 220,	2		
30031100000000,	200222121,220222020,	221 220,	3		
30031300000000,	200222221,220222220,	220 220,	4		
	100111111,011000000,	011 100,		1	
	100111111,011000000,	011 000,		2	
	100111111,010000000,	011 100,		3	
	100111111,010000000,	011 000,		4	
	100110111,010001000,	001 010,		5	general solutions
	100111101,010000000,	010 100,		6	
	100111111,001000000,	011 100,		7	
	100111111,001000000,	011 000,		8	
	100111111,000000000,	011 100,		9	
	100111111,000000000,	011 000,		10	
11111221111200,	100122222,012022222,	012 102,	1		
11111231111200,	100222122,011222022,	012 002,	2		
11111231111300,	100222122,001222022,	011 000,	3		
11111321111200,	100122222,001022222,	011 100,	4		
11111321111300,	100122222,000022222,	021 120,	5		
omitted					


```

33031212313313, 200221121,210220020, 000 100, 21
33031213313133, 200220121,221220020, 000 000, 22
33031213313213, 200221121,221220020, 000 000, 23
33031213313223, 200220121,221221020, 000 000, 24
33031223113111, 100120121,021021020, 001 000, 25
33031312312213, 200221121,201220020, 000 110, 26
33031312313213, 200222121,201222020, 000 100, 27
33031312313313, 200222121,200222020, 020 120, 28
33031313313313, 200222121,220222020, 020 020, 29
33032312311213, 200222211,201222200, 010 100, 30
33032312312213, 200220211,201221200, 000 110, 31
33033212312133, 200220221,212220220, 000 100, 32
33033312312213, 200220221,201220220, 000 110, 33
10011220000000, 100122222,012022222, 022 122, 1
10011230000000, 100122122,021022022, 022 022, 2
10011320000000, 100122122,002022022, 022 122, 3
10011330000000, 100222122,020222022, 022 022, 4
20011120000000, 100212222,022202222, 202 212, 1
20011130000000, 100221212,022220202, 202 202, 2
20011230000000, 100120212,022021202, 202 202, 3
20011330000000, 100220112,022220002, 202 202, 4
20022110000000, 200112222,212002222, 212 202, 5
20023110000000, 200121222,201020222, 212 202, 6
20023130000000, 200221202,221220202, 202 202, 7
20033110000000, 200122221,200022220, 212 202, 8
20033130000000, 200221201,220220200, 202 202, 9
30011130000000, 100222221,022222220, 220 220, 1
30022110000000, 200212122,211202022, 221 220, 2
30032110000000, 200222112,201222002, 221 220, 3
30033110000000, 200222121,220222020, 221 220, 4

```

IMP(A,B) NEG(A)

```

A 111222333 123 P
B 123123123

```

```

TRUTH VALUE
122111111 211 1
122111111 311 2
123111111 211 3
123111111 311 4
123112111 321 5
123111131 213 6
132111111 211 7
132111111 311 8
133111111 211 9
133111111 311 10

```

```

1=IMP(NEG(IMP(X,NEG(X))),
    IMP(Y,IMP(IMP(Z,IMP(NEG(X),X)),
        NEG(IMP(Y,IMP(X,NEG(X))))))),

```

```

1=IMP(NEG(IMP(X,NEG(X))),
    IMP(IMP(IMP(Z,IMP(NEG(X),X)),
        NEG(IMP(Y,IMP(X,NEG(X))))),Y)), (10.7)

```

```

1=IMP(NEG(X2),
    IMP(Y,NEG(IMP(IMP(X2,NEG(Y)),NEG(Y)))),

```

```

1=IMP(NEG(X2),
    IMP(NEG(IMP(IMP(X2,NEG(Y)),NEG(Y))),Y)),

```

```

X2=IMP(IMP(NEG(X),X),NEG(IMP(X,NEG(X))), (10.8)

```

```

1=IMP(IMP(IMP(X,NEG(X)),
    IMP(NEG(X),X)),
    IMP(NEG(X),X)), (11.2.1)

```

Next we should confirm F123 as the canonical form of $F(X)$ for $P = 1 \sim 10$.

At first, the following expressions

$$7X1 \supset (C1 \supset F13), \quad 7X1 \supset (F13 \supset C1), \quad (10.7)$$

$$7X3 \supset (C3 \supset F13), \quad 7X3 \supset (F13 \supset C3), \quad (10.7')$$

$$7X2 \supset (C2 \supset F2), \quad 7X2 \supset (F2 \supset C2) \quad (10.8)$$

are to be assumed as supplementary axioms while Eqs. (10.7') are unnecessary, because Eqs. (10.7') are equivalent to Eqs. (10.7) as in Eqs. (10.4).

They are confirmed to satisfy the general solutions $P = 1 \sim 10$ using a computer, as shown by Eqs. (10.7) \sim (10.8) in Table 9.1, where

$$Y \equiv C1 \text{ and } Z \equiv C3 \text{ for Eqs. (10.7),}$$

$$Y \equiv C2 \text{ for Eqs. (10.8).}$$

Then, combining Eqs. (10.7) with Eqs. (10.8) as in Eq. (10.5), we have,

$$7Xn \supset (Cn \supset F123), \quad 7Xn \supset (F123 \supset Cn) \quad (10.9)$$

for $n = 1, 2, 3$.

Eqs. (10.7) \sim (10.9) lead us to the following tautologies for $n = 1, 2, 3$,

$$7Xn \supset ((F(X) \supset Cn) \supset (F(X) \supset F123)),$$

$$7Xn \supset ((Cn \supset F(X)) \supset (F123 \supset F(X))), \quad (10.10)$$

which mean "F123 is equivalent to $F(X)$ ".

Otherwise, Eqs. (10.7) \sim (10.10) are also derived from the axioms (6.1) \sim (6.4) as shown in App. 1 (10.7.1), (10.7.8), (10.8.1), (10.8.3).

Then we may conclude "The expression F123 is a canonical form of a function $F(X)$ " provided that $7Xn = 1$ for $n = 1, 2, 3$ covers all the truth-values of X , satisfying the axiom set (6.0) \sim (6.4). It will be clarified in the next paragraph.

As for a two-variable function $F(X, Y)$, we may treat Cn as a given function of Y .

11 Fittest Number of Truth-Values

For simplicity, let us define the three-valued equivalence \equiv as follows:

$$(X \equiv Y) \equiv (X \supset Y) \cap (Y \supset X). \quad (11.1)$$

Theorem 1⁰ "7X1 = 1" does not coexist with

$$7X3 = 1.$$

Proof. Theorem 1⁰ is expressed by Eq. (11.2)

$$7(7X1 \cap 7X3) \quad (11.2)$$

$$\equiv (X1 \cup X3)$$

$$\equiv ((X1 \supset X3) \supset X3). \quad (11.2') \text{ (see (2.1))}$$

Then, assuming Eq. (11.2') as a supplementary axiom, we can confirm

$$1 = \text{Eq. (11.2')}$$

for $P = 1 \sim 10$, using a computer, as shown in Table 9.1 [see App. 1 (11.2.1)]

Theorem 2⁰ If "one of 7X1, 7X2 and 7X3 is equal to 1" and "Another of them is not equal to 1", then "The remainder of them is not equal to 1".

Proof. $[X1 \supset (7X3 \supset X2)] \supset [7X3 \supset (X1 \supset X2)]$

$$\supset [X1 \supset (7X2 \supset X3)]$$

$$\supset [7X2 \supset (X1 \supset X3)] \quad (11.3)$$

$$\supset [7X2 \supset (7X3 \supset 7X1)] \quad (11.3')$$

$$\supset [X1 \supset (7X3 \supset (X1 \supset 7X3))]$$

$$\supset [X1 \supset 1]$$

$$\supset 1.$$

$$[X2 \supset (7X1 \supset X3)] \supset 1.$$

$$\therefore (10.7.2), (10.7.3) \vdash [7X1 \supset X3]. \quad (11.4')$$

Eq. (11.3') seems to include the following case

$$7X2 = 7X3 = 7X1 = 1, \quad (11.5)$$

while "Eq. (11.5) does not exist" according to Theorem 1⁰.

Theorem 3⁰ If "Two of 7X1, 7X2 and 7X3 are not equal to 1", then "The remainder of them is equal to 1".

Proof. $[X1 \supset (X3 \supset 7X2)] \supset [X1 \supset (X2 \supset 7X3)]$

$$\supset [X3 \supset (X2 \supset 7X1)]$$

$$\supset [X1 \supset (X3 \supset 7(X3 \supset 7X1))]$$

$$\supset [X1 \supset ((X3 \supset 7X1) \supset 7X3)]$$

$$\supset [(X3 \supset 7X1) \supset (X1 \supset 7X3)]$$

$$\supset [(X3 \supset 7X1) \supset (X3 \supset 7X1)]$$

$$\supset 1.$$

$$(11.6)$$

The following conclusion is derived from the above-mentioned theorems 1⁰, 2⁰, 3⁰.

Theorem 1 2 3⁰ The domain of X is divided into three subdomains 7X1, 7X2 and 7X3, where "One of is equal to 1", and "The remaining two of them are not equal to 1".

Then $7Xn = 1$ for $n = 1, 2, 3$ covers all the truth-values of X , satisfying the axiom set (6.0) \sim (6.4).

Of course if we divide the subdomain $X = 3$ into two parts such as "3" and "4", then we will have 4-valued logic, but, as shown in Tables (9.1) and (10.1), there are two typical solutions ($P = 5, 6$), which keep the one to one correspondence between $7Xn = 1$ and X .

Then the fittest number of truth-values for the axiom set (6.0) \sim (6.8) is regarded as "3".

12 Conclusion

In this paper, undefined operators and axioms for many-valued logic are treated as unknown logical functions and logical equations respectively. Solving those equations by a computer, the general solutions give the general truth tables for the undefined operators. As effective applications of this method, canonical forms of logical functions which satisfy the general truth tables are derived and the fittest number of the truth-values for the given axiom set is also determined in relation to the construction of the canonical forms.

Appendix

1 Proof of Tautologies

$$[7X1 \supset (C1 \supset F13)]$$

$$\supset [7X1 \supset (C1 \supset ((C3 \supset X3) \supset 7(C1 \supset X1)))]$$

$$\supset [7X1 \supset ((C3 \supset X3) \supset (C1 \supset 7(C1 \supset X1)))]$$

$$\supset [(C3 \supset X3) \supset (7X1 \supset (C1 \supset 7(C1 \supset X1)))]$$

$$\supset [(C3 \supset X3) \supset (7X1 \supset ((C1 \supset X1) \supset 7C1))]$$

$$\supset [(C3 \supset X3) \supset ((C1 \supset X1) \supset (7X1 \supset 7C1))]$$

$$\supset [(C3 \supset X3) \supset ((C1 \supset X1) \supset (C1 \supset X1))]$$

$$\supset [(C3 \supset X3) \supset 1]$$

$$\supset 1.$$

$$(10.7.1)$$

$$[7X1 \supset X] \supset [7(X \supset 7X) \supset X] \supset [7X \supset (X \supset 7X)] \supset 1.$$

$$(10.7.2)$$

$$[X \supset X] \supset [X \supset (7X \supset X)] \supset 1.$$

$$(10.7.3)$$

$$[X3 \supset (C3 \supset X3)] \supset 1.$$

$$(10.7.4)$$

$$(10.7.2), (10.7.3), (10.7.4) \vdash [7X1 \supset (C3 \supset X3)] \supset 1.$$

$$(10.7.5)$$

$$1 \supset [((C3 \supset X3) \supset 7(C1 \supset X1)) \supset ((C3 \supset X3) \supset 7(C1 \supset X1))]$$

$$\supset [(C3 \supset X3) \supset (((C3 \supset X3) \supset 7(C1 \supset X1)) \supset 7(C1 \supset X1))].$$

$$(10.7.6)$$

$$[7(C1 \supset X1) \supset C1] \supset [7C1 \supset (C1 \supset X1)]$$

$$\supset [7C1 \supset (7X1 \supset 7C1)] \supset 1 \quad (10.7.7)$$

(10.7.5), (10.7.6), (10.7.7) \vdash
 $[7X1 \supset ((C3 \supset X3) \supset 7(C1 \supset X1)) \supset C1]$. (10.7.8)
 $[7X2 \supset (C2 \supset F2)] \mathbf{2} [7X2 \supset (C2 \supset 7((X2 \supset 7C2) \supset 7C2))]$
 $\mathbf{2} [7X2 \supset ((X2 \supset 7C2) \supset 7C2)]$
 $\mathbf{2} [(X2 \supset 7C2) \supset (7X2 \supset 7C2)]$
 $\mathbf{2} \mathbf{1}$. (10.8.1)
 $\therefore [7A \supset (A \supset B)] \mathbf{2} [7A \supset (7B \supset 7A)] \mathbf{2} \mathbf{1}$
 $\therefore [((A \supset B) \supset C) \supset (7A \supset C)] \mathbf{2} \mathbf{1}$. (10.8.2)
 $[7X2 \supset (F2 \supset C2)]$
 $\mathbf{2} [7X2 \supset (7((X2 \supset 7C2) \supset 7C2) \supset C2)]$
 $\mathbf{2} [7X2 \supset (7C2 \supset ((X2 \supset 7C2) \supset 7C2))]$
 $\mathbf{2} [7X2 \supset \mathbf{1}]$
 $\mathbf{2} \mathbf{1}$. (10.8.3)
 $[(X1 \supset X3) \supset X3] \mathbf{2} [((X \supset 7X) \supset (7X \supset X)) \supset (7X \supset X)]$
 $\mathbf{2} [(7X \supset ((X \supset 7X) \supset X)) \supset (7X \supset X)]$
 $\mathbf{2} \mathbf{1}$. (11.2.1)
 $\therefore [(Y \supset ((Z \supset 7Z) \supset Z)) \supset (Y \supset Z)] \mathbf{2} \mathbf{1}$. (11.2.2)²

2 References

- (1) Wajsberg, Mordchaj, at p.355 in Nicholas Rescher; Many-Valued Logic, McGraw-Hill (1969).
- (2) Wajsberg (1931), tr. in S. McCall (ed.), Polish Logic; 1920-1939 (Oxford, 1967), 12 at p.267.
- (3) Motinori Goto; Various Types of General Solutions of Many-Valued Logical Equations with Many Variables. Bulletin of the Electrotechnical Laboratory (ETL), vol.20, No.9 (Tokyo, Japan, Sept. 1956).

- (4) M. Goto, Y. Komamiya, etc.: Theory and Structure of the Automatic Relay Computer ETL, Mark II. Researches of ETL. No. 556 (ditto).
- (5) M. Goto, S. Kao, T. Ninomiya; Determination of Many-Valued Truth Tables for Undefined Operators in Axioms by a Computer and Their Applications. Proceedings of the seventh international symposium on multiple-valued logic. May, 1977. pp.20 - 28.

Table of Errata in (5)

page	line, etc.	for	read
21	left (7.4)	$\dots, X), X), X) \models \mathbf{1}$	$\dots, X), X) \models \mathbf{1}$
21	left 2 above bottom	I_{xy}^j	I_{yx}^j
21	right 11	1 8.0)	(8.0)
21	right (9.6)	IBC = 14,15	IBC \neq 14,15
23	right 9	eiht	eight
23	right 12 above bottom	give	given

VECTOR REPRESENTATION OF SWITCHING AND THREE-VALUED FUNCTIONS[†]

Ytzhak Levendel

Bell Telephone Laboratories
Naperville, Illinois
60540

Melvin A. Breuer

Departments of Electrical Engineering
and Computer Science
University of Southern California
Los Angeles, California 90007

ABSTRACT

In this paper we present a new vector representation for switching functions, where two or three-valued logic systems are employed. Using this representation, rules for carrying out numerous operations are presented such as vector AND, OR and complementation; Boolean difference; vector reduction; and substitution of 0, 1 or u for the variables. Hence complex operations classically carried out over large Boolean expressions can now be more easily carried out using simple algorithms which operate over the components of these new vector representations.

I. INTRODUCTION

Many procedures related to the design and analysis of digital circuits employ algebraic techniques. These techniques often require the generation and manipulation of large Boolean expressions. An example of this situation occurs when the Boolean difference¹ procedure is used for test generation. Other computations require the generation of prime implicants and minimal covers, the latter problem being of exponential complexity.

As a possible means of alleviating this situation we have studied a new representation of switching (binary) and three-valued expressions. These results will be the subject of this paper.

The application of these representations reside in the modeling of sequential circuits. Given a sequential circuit with memory elements y_1, y_2, \dots, y_n , and input lines x_1, x_2, \dots, x_n , the excitation variables (inputs to the memory elements) are given by functions of the form

$$f(\underline{y}, \underline{x}) = f(y_1, y_2, \dots, y_n, x_1, x_2, \dots, x_n).$$

The vector \underline{x} may be considered as a parametric vector in which case f becomes a function of n variables, denoted by \underline{y} .

Section II proposes a vector representation of length 2^n for parametric switching functions of n variables and gives mathematical rules for the manipulation of such expressions.

[†]This work was supported by the Office of Naval Research under contract number N00014-67-A-069-0019 (NR 048-299) and by the Joint Services Electronics Program through AFOSR under contract number F44620-76-C-0061.

When one uses a three-valued logic system and substitutes $y_i = u$ (unknown) for all i , the function f becomes $f'(u, \underline{x})$. A representation for f' was proposed by Chappell⁴ using two expressions, which were called the zero and one components. It is interesting to consider the case where only some of the y_i are replaced by u , and f thus takes the form $f''(y_1, y_2, \dots, y_i, u, \underline{x})$. In Section III we present a representation for f'' using two vectors of length 2^n . Applications of these vector representations to generating fault detection tests and synchronizing sequences are given in Levendel.³

II. VECTOR BOOLEAN ALGEBRA

Notation and Basic Results

Given n Boolean variables y_1, y_2, \dots, y_n , the minterm vector corresponding to these variables is a column vector \underline{m} of length 2^n . It is composed of all the minterms over y_1, y_2, \dots, y_n written in a canonical order. This order is as follows

$$\underline{m} = (\bar{y}_n \bar{y}_{n-1} \dots \bar{y}_1, \bar{y}_n \bar{y}_{n-1} \dots \bar{y}_2 y_1, \bar{y}_n \bar{y}_{n-1} \dots y_2 \bar{y}_1, \bar{y}_n \bar{y}_{n-1} \dots y_2 y_1, \dots, y_n y_{n-1} \dots y_2 y_1)$$

or

$$\underline{m} = (m_0, m_1, \dots, m_p) \quad \text{where } p = 2^n - 1.$$

Consider a function $f(\underline{y}, \underline{x})$, where $\underline{y} = (y_1, y_2, \dots, y_n)$ and $\underline{x} = (x_1, x_2, \dots, x_n)$.

Theorem 1: It is possible to write

$$f(\underline{y}, \underline{x}) = \underline{q} \cdot \underline{m} = \underline{q}(\underline{x}) \cdot \underline{m}(\underline{y}) \quad (1)$$

where the dot operator represents a scalar product. \underline{q} is a row vector of size 2^n and is called the characteristic vector of $f(\underline{y}, \underline{x})$.

Proof: The function $f(\underline{y}, \underline{x})$ can be written in a sum of minterm form as

$$f(\underline{y}, \underline{x}) = \sum_{i=0}^p f_i(\underline{x}) m_i$$

where $f_i(\underline{x})$ is the value of $f(\underline{y}, \underline{x})$ obtained by assigning to \underline{y} the values for which $m_i = 1$. By selecting the components of \underline{q} using the assignment $\phi_i = f_i(\underline{x})$ the theorem is proven. ■

Let S be the space of all characteristic vectors of dimension 2^n . Given two vectors \underline{u}_1 and \underline{u}_2 which belong to S , let us define

1. a zero vector $\underline{0} = (0, 0, \dots, 0)$ of dimension 2^n .
2. a one vector $\underline{1} = (1, 1, \dots, 1)$ of dimension 2^n .
3. an AND operation denoted by " \cdot " such that

$$\underline{u}_1 \cdot \underline{u}_2 = (\varphi_{10} \cdot \varphi_{20}, \varphi_{11} \cdot \varphi_{21}, \dots, \varphi_{1p} \cdot \varphi_{2p})$$
4. an OR operation denoted by " $+$ " such that

$$\underline{u}_1 + \underline{u}_2 = (\varphi_{10} + \varphi_{20}, \varphi_{11} + \varphi_{21}, \dots, \varphi_{1p} + \varphi_{2p})$$
5. a complement operation denoted by " $\bar{}$ " such that

$$\bar{\underline{u}}_1 = (\bar{\varphi}_{10}, \bar{\varphi}_{11}, \dots, \bar{\varphi}_{1p})$$

Theorem 2: The system $(S, \underline{0}, \underline{1}, +, \cdot, \bar{})$ forms a Boolean algebra.

Proof: It can be easily shown that this system satisfies Huntington's postulates. ■

Example 1: Let us consider functions of two variables y_1 and y_2 and two parameters x_1 and x_2 . Then

$$\underline{m} = (\bar{y}_2 \bar{y}_1, \bar{y}_2 y_1, y_2 \bar{y}_1, y_2 y_1)$$

and

$$\begin{aligned} y_1 &= (0, 1, 0, 1) \cdot \underline{m} \\ \bar{y}_2 &= (1, 1, 0, 0) \cdot \underline{m} \\ \bar{y}_1 &= (1, 0, 1, 0) \cdot \underline{m} \\ \bar{x}_1 &= (\bar{x}_1, \bar{x}_1, \bar{x}_1, \bar{x}_1) \cdot \underline{m} \\ \bar{x}_1 + \bar{y}_2 &= (1, 1, \bar{x}_1, \bar{x}_1) \cdot \underline{m} \end{aligned}$$

Note: It is possible to eliminate \underline{m} for notational simplicity. In fact, \underline{m} represents a base and \underline{u} a vector of coordinates.

Theorem 3: Given r functions of n variables y_1, y_2, \dots, y_n , denoted by

$$z_j = f_j(\underline{y}, \underline{x}) = \underline{u}_j \cdot \underline{m} \quad j = 0, 1, \dots, r-1$$

and a function f of r variables $f(z_0, z_1, \dots, z_{r-1})$, then

$$\begin{aligned} f(z_0, z_1, \dots, z_{r-1}) &= f(\underline{u}_0, \underline{u}_1, \dots, \underline{u}_{r-1}) \cdot \underline{m} \\ &= (f(\varphi_{00}, \varphi_{10}, \dots, \varphi_{r-1,0}), \\ &\quad f(\varphi_{11}, \varphi_{21}, \dots, \varphi_{r-1,1}), \dots, f(\varphi_{0p}, \varphi_{1p}, \dots, \varphi_{r-1,p})) \\ &\quad \cdot \underline{m} \end{aligned}$$

Proof: The proof follows from the fact that the vector operations $+$, \cdot and complement have been defined component by component. Since any Boolean function can be expressed in terms of these basic operators, the theorem follows. ■

Example 2: Given

$$\begin{aligned} z_0 &= (\varphi_{00}, \varphi_{01}, \varphi_{02}, \varphi_{03}) \cdot \underline{m} \\ z_1 &= (\varphi_{10}, \varphi_{11}, \varphi_{12}, \varphi_{13}) \cdot \underline{m} \\ z_2 &= (\varphi_{20}, \varphi_{21}, \varphi_{22}, \varphi_{23}) \cdot \underline{m} \end{aligned}$$

then

$$\begin{aligned} z_0 z_1 + \bar{z}_2 &= (\varphi_{00} \varphi_{10} + \bar{\varphi}_{20}, \varphi_{01} \varphi_{11} + \bar{\varphi}_{21}, \varphi_{02} \varphi_{12} + \bar{\varphi}_{22}, \\ &\quad \varphi_{03} \varphi_{13} + \bar{\varphi}_{23}) \cdot \underline{m} \end{aligned}$$

Single Variable Manipulation

Consider $f(y_1, y_2, \dots, y_i, \dots, y_n, \underline{x}) = \underline{u} \cdot \underline{m}$ and let us define \underline{u}_i^* such that

$$f(y_1, \dots, \bar{y}_i, \dots, y_n, \underline{x}) = \underline{u}_i^* \cdot \underline{m}$$

The Boolean difference¹ with respect to the variable y_i is defined as

$$\begin{aligned} \frac{df(\underline{y}, \underline{x})}{dy_i} &= f(y_1, \dots, y_i, \dots, y_n, \underline{x}) \\ &\quad \oplus f(y_1, \dots, \bar{y}_i, \dots, y_n, \underline{x}) \end{aligned} \quad (2)$$

Theorem 4: We have that

$$df(\underline{y}, \underline{x})/dy_i = (\underline{u} \oplus \underline{u}_i^*) \cdot \underline{m}$$

Given a characteristic vector

$$\underline{u} = (\varphi_0, \varphi_1, \dots, \varphi_i, \dots, \varphi_j, \dots, \varphi_p),$$

the σ -operator is defined as follows:

$$\sigma_i^j \underline{u} = (\varphi_0, \varphi_1, \dots, \varphi_j, \dots, \varphi_i, \dots, \varphi_p)$$

This operator interchanges the i^{th} and j^{th} components in \underline{u} .

The following procedure computes \underline{u}_i^* , given \underline{u} .

Procedure 1: Generation of \underline{u}_i^* .

1. Set $t = 0$ (t is of dimension 2^n).
Set $\underline{u}_i^* = \underline{u}$.
Set $j = 0$.
2. If $t_j = 1$ then go to 3, else continue.
Set $\underline{u}_i^* = \sigma_j^{i+1} \underline{u}_i^*$.
Set $t_{j+1} = 1$.
3. Set $j = j+1$.
If $j = p$ then stop, else go to 2.

Example 3: Consider the function

$$f(\underline{y}, \underline{x}) = \bar{x}_1 + \bar{x}_2 + y_1 \bar{y}_2 + \bar{y}_1 y_2 = (\bar{x}_1 + x_2, 1, 1, \bar{x}_1 + \bar{x}_2) \cdot \underline{m}$$

We want to compute $df(\underline{y}, \underline{x})/dy_2$.

Given

$$\underline{w} = (\bar{x}_1 + \bar{x}_2, 1, 1, \bar{x}_1 + \bar{x}_2)$$

we want to obtain \underline{w}_2^* by using Procedure 1.

Step 1 produces

$$\underline{t} = \underline{0} \quad \underline{w}_2^* = \underline{w} \quad j = 0.$$

Step 2 produces

$$\underline{w}_2^* = (1, 1, \bar{x}_1 + \bar{x}_2, \bar{x}_1 + \bar{x}_2)$$

$$\underline{t} = (0, 0, 1, 0).$$

Step 3 produces

$$j = 1.$$

Step 4 produces

$$\underline{w}_2^* = (1, \bar{x}_1 + \bar{x}_2, \bar{x}_1 + \bar{x}_2, 1)$$

$$\underline{t} = (0, 0, 1, 1)$$

and the procedure stops. \underline{w}_2^* is the correct expression.

By definition of the Boolean difference¹

$$df(\underline{y}, \underline{x})/dy_2 = f(y_1, y_2, \underline{x}) \oplus f(y_1, \bar{y}_2, \underline{x})$$

$$= (\underline{w} \oplus \underline{w}_2^*) \cdot \underline{m}$$

$$= (x_1 x_2, x_1 x_2, x_1 x_2, x_1 x_2) \cdot \underline{m} = x_1 x_2. \blacksquare$$

Given a characteristic vector

$$\underline{w} = (w_0, w_1, \dots, w_i, \dots, w_j, \dots, w_p),$$

the τ -operator is defined as follows:

$$\tau^j \underline{w} = (w_0, w_1, \dots, w_i, \dots, w_j, \dots, w_p).$$

Given $f(y_1, \dots, y_i, \dots, y_n, \underline{x})$ assume one desires to obtain \underline{w}_i^* such that

$$\underline{w}_i^* \cdot \underline{m} = f(y_1, \dots, \alpha, \dots, y_n, \underline{x}) \text{ for } \alpha = 0, 1.$$

If we use the following definition of Boolean difference

$$df(\underline{y}, \underline{x})/dy_i = f(y_1, \dots, y_{i-1}, 0, y_{i+1}, \dots, y_n)$$

$$\oplus f(y_1, \dots, y_{i-1}, 1, y_{i+1}, \dots, y_n)$$

and by a proof similar to that of Theorem 4 we obtain the following result.

Theorem 5: We have that

$$df(\underline{y}, \underline{x})/dy_i = (\underline{w}_i^0 \oplus \underline{w}_i^1) \cdot \underline{m}.$$

The next procedure computes \underline{w}_i^* .

Procedure 2: Computation of \underline{w}_i^* .

1. Set $t = 0$.

Set $\underline{w}_i^* = \underline{w}$.

Set $j = 0$.

2. If $t_j = 1$, then go to 3, else continue.

If $a = 1$, then set $\underline{w}_i^* = \tau_{j+a}^j \underline{w}_i^*$.

If $a = 0$, then set $\underline{w}_i^* = \tau_{j+1}^j \underline{w}_i^*$.

Set $t_{j+1} = 1$.

3. Set $j = j+1$.

If $j = p$ then stop, else go to 2. \blacksquare

The preceding procedure generates a vector \underline{w}_i^* which is independent of y_i . This implies that it should be possible to reduce \underline{w}_i^* to a vector \underline{w}' of dimension 2^{n-1} . This will be done by the following procedure.

Procedure 3: Generation of \underline{w}' from \underline{w}_i^* .

1. Set $t=0$, $j=0$, $k=0$ and $\underline{w}' = *$ (unspecified vector of dimension 2^{n-1}). (Let an element of \underline{w}' be w'_k , and an element of \underline{w}_i^* be w_h .)

2. If $t_j = 1$, then go to 3 else continue.

Compute $h = j + ai$ and set $w'_k = w_h$.

Set $t_{j+1} = 1$.

Set $k = k+1$.

3. Set $j = j+1$.

If $j = p$ then stop, else go to 2. \blacksquare

Procedure 3 can be used iteratively when we assign several y_i variables to constant values.

Theorem 6: Given a characteristic vector $\underline{w}(\underline{x})$ corresponding to a function $f(\underline{y}, \underline{x})$, then $f(\underline{y}, \underline{x})$ is independent of \underline{y} if and only if

$$w_0 = w_1 = w_2 = \dots = w_p. \blacksquare$$

Example 4: Consider the characteristic vector

$$\underline{w} = (\bar{x}_1 + \bar{x}_2, 1, \bar{x}_1 + \bar{x}_2)$$

corresponding to the function $\bar{x}_1 + \bar{x}_2 + y_1 \bar{y}_2 + \bar{y}_1 y_2$. Then it is necessary to have $\bar{x}_1 + \bar{x}_2 = 1$ in order to have independence of \underline{y} . This implies

$$x_1 = 0 \quad \text{or} \quad x_2 = 0. \blacksquare$$

III. THREE VALUED VECTOR ALGEBRA

Notation and Results

Consider the characteristic vectors of dimension 2^n . Each component of such a vector may take the values 0, 1 or u. An assignment of values to the components is called a valuation. The characteristic vectors, the valuations of which belong to $\{0, 1, u\}$ form a space, denoted by S^3 . We define two spaces S' and S'' such that for each vector $\underline{w} \in S^3$ there are two vectors $\underline{w}' \in S'$ and $\underline{w}'' \in S''$ obtained according to the following rules:

1. Given $\varphi_1 = 0$ then $\varphi_1^f = 0$ and $\varphi_1^r = 1$.
2. Given $\varphi_1 = 1$ then $\varphi_1^f = 1$ and $\varphi_1^r = 0$.
3. Given $\varphi_1 = u$ then $\varphi_1^f = 0$ and $\varphi_1^r = 0$.

Note that the components of each vector in the spaces S^r and S^f are binary.

From this coding, the vectors $\underline{0}$, $\underline{1}$ and \underline{u} in S^3 take on the following form:

- a. For $\underline{0} \in S^3$, then $\underline{0}^r = (0, 0, \dots, 0)$
 $\underline{0}^f = (1, 1, \dots, 1)$.
- b. For $\underline{1} \in S^3$, then $\underline{1}^r = (1, 1, \dots, 1)$
 $\underline{1}^f = (0, 0, \dots, 0)$.
- c. For $\underline{u} \in S^3$, then $\underline{u}^r = (0, 0, \dots, 0)$
 $\underline{u}^f = (0, 0, \dots, 0)$.

This coding scheme was first introduced by Chappell⁵, where it was restricted to scalars only.

The AND, OR and COMPLEMENT operators are defined in Table 1. Vector logical operations are carried out according to this table, component by component.

AND	0	1	u
0	0	0	0
1	0	1	u
u	0	u	u

(a)

OR	0	1	u
0	0	1	u
1	1	1	1
u	u	1	u

(b)

	COMPLEMENT
0	1
1	0
u	u

(c)

TABLE 1

Theorem 7: Given the vectors $\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5 \in S^3$, the following vector operations exist:

1. If $\varphi_3 = \varphi_1 \cdot \varphi_2$, then $\varphi_3^r = \varphi_1^r \cdot \varphi_2^r$
and $\varphi_3^f = \varphi_1^f + \varphi_2^f$.
2. If $\varphi_4 = \varphi_1 + \varphi_2$, then $\varphi_4^r = \varphi_1^r + \varphi_2^r$
and $\varphi_4^f = \varphi_1^f \cdot \varphi_2^f$.
3. If $\varphi_5 = \bar{\varphi}_1$, then $\varphi_5^r = \varphi_1^f$
and $\varphi_5^f = \varphi_1^r$.

Corollary 1: The superscript F can be considered as an operator similar to the complement and it satisfies "DeMorgan Rules"

1. $(\varphi_1 \cdot \varphi_2)^f = \varphi_1^f + \varphi_2^f$
2. $(\varphi_1 + \varphi_2)^f = \varphi_1^f \cdot \varphi_2^f$.

Given a function $\varphi = f(\varphi_1, \varphi_2, \dots, \varphi_n)$, the transposed function of f , denoted by f_t , is obtained from f by the following transformation

AND \rightarrow AND
OR \rightarrow OR
COMPLEMENT \rightarrow "F" OPERATOR.

Corollary 2: We have

$$\varphi^r = f_t(\varphi_1^r, \varphi_2^r, \dots, \varphi_n^r)$$

and

$$\varphi^f = [f_t(\varphi_1^f, \varphi_2^f, \dots, \varphi_n^f)]^f.$$

Corollary 3: Whenever the variables can only take the values 0 and 1, then

$$\varphi^r = \varphi \quad \text{and} \quad \varphi^f = \bar{\varphi}.$$

Example 5: For the vector $\varphi = (\bar{x}_1 + \bar{x}_2, u, 0, 1)$ we will compute φ^r and φ^f . The computation is done component by component. First we compute φ_0^r and φ_0^f (see Table 2).

x_1	x_2	x_1^r	x_1^f	x_2^r	x_2^f	φ_0^r	φ_0^f	$\bar{x}_1 + \bar{x}_2$
0	0	0	1	0	1	1	0	1
0	1	0	1	1	0	1	0	1
0	u	0	1	0	0	1	0	1
1	0	1	0	0	1	1	0	1
1	1	1	0	1	0	0	1	0
1	u	1	0	0	0	0	0	u
u	0	0	0	0	1	1	0	1
u	1	0	0	1	0	0	0	u
u	u	0	0	0	0	0	0	u
		other combinations				-	-	

Table 2. Truth Table for Example 5.

Extracting φ_0^r and φ_0^f from the truth table we obtain

$$\varphi_0^r = x_1^r + x_2^r$$

$$\varphi_0^f = x_1^f x_2^f.$$

We also have

$$\begin{aligned} \varphi_1^r &= 0 & \varphi_2^r &= 0 & \varphi_3^r &= 1 \\ \varphi_1^f &= 0 & \varphi_2^f &= 1 & \varphi_3^f &= 0. \end{aligned}$$

Finally we obtain

$$\varphi^r = (x_1^r + x_2^r, 0, 0, 1)$$

$$\varphi^f = (x_1^f x_1^f, 0, 1, 0)$$

and if we assume that x_1 and x_2 are evaluated over $\{0, 1\}$, then

and $\underline{w}^T = (\bar{x}_1 + \bar{x}_2, 0, 0, 1)$
 $\underline{w}^F = (x_1 x_2, 0, 1, 0).$ ■

We can obtain the same result by applying Theorem 7 to each component.

Example 6: Let us compute \underline{w}^T and \underline{w}^F for the function

$$xy_1 + u \quad \text{where} \quad x, y_1 \in \{0, 1\}.$$

We have

$$\underline{m} = (\bar{y}_1, y_1)$$

and

$$\underline{x}^T = (x, x) \quad \underline{y}_1^T = (0, 1) \quad \underline{u}^T = (0, 0)$$

$$\underline{x}^F = (\bar{x}, \bar{x}) \quad \underline{y}_1^F = (1, 0) \quad \underline{u}^F = (0, 0).$$

Then

$$\underline{w}^T = \underline{x}^T \underline{y}_1^T + \underline{u}^T = (0, x)$$

and

$$\underline{w}^F = (\underline{x}^T \underline{y}_1^F + \underline{u}^F)^F$$

$$= \underline{x}^F \underline{u}^F + \underline{y}_1^F \underline{u}^F + \underline{y}_1^F \underline{u}^F = (0, 0).$$
 ■

Single Variable Manipulation

Given \underline{w}^T and \underline{w}^F the following is a procedure for calculating \underline{w}_i^{*T} and \underline{w}_i^{*F} , i.e., it allows us to complement the variable y_i .

Procedure 4: Complementation of variable y_i .

1. Apply Procedure 1 to \underline{w}^T and obtain \underline{w}_i^{*T} .
2. Apply Procedure 1 to \underline{w}^F and obtain \underline{w}_i^{*F} . ■

Procedure 5: Assignment of $a = 0, 1$ to y_i .

1. Apply Procedure 2 to \underline{w}^T and obtain \underline{w}_i^{*T} .
Apply Procedure 3 to \underline{w}_i^{*T} and obtain \underline{w}^{*T} .
2. Apply Procedure 2 to \underline{w}^F and obtain \underline{w}_i^{*F} .
Apply Procedure 3 to \underline{w}_i^{*F} and obtain \underline{w}^{*F} . ■

The result is \underline{w}^{*T} and \underline{w}^{*F} which have dimension 2^{n-1} .

Given \underline{w} , the following procedure assigns the value u to the variable y_i . The result will be \underline{w}^{*T} and \underline{w}^{*F} .

Procedure 6: Assignment of u to y_i .

1. Apply Procedure 2 to \underline{w} and obtain \underline{w}_i^0 and \underline{w}_i^1 .
2. Compute $\underline{w}_1 = \underline{w}_i^0 \cdot \underline{w}_i^1$
and $\underline{w}_2 = \underline{w}_i^0 + \underline{w}_i^1$.
3. Apply Procedure 3 to \underline{w}_1 and \underline{w}_2 and obtain \underline{w}'_1 and \underline{w}'_2 .
4. Then $\underline{w}^{*T} = \underline{w}'_1$ and $\underline{w}^{*F} = \underline{w}'_2$. ■

Example 7: Given $\bar{x}y_1 + y_1\bar{y}_2$, we want to assign $y_2 = u$. We have

$$\underline{w} = (\bar{x}, 1, \bar{x}, 0).$$

We obtain

$$\underline{w}_2^0 = (\bar{x}, 1, \bar{x}, 1)$$

$$\underline{w}_2^1 = (\bar{x}, 0, \bar{x}, 0)$$

and

$$\underline{w}_1 = \underline{w}_2^0 \cdot \underline{w}_2^1 = (\bar{x}, 0, \bar{x}, 0)$$

$$\underline{w}_2 = \underline{w}_2^0 + \underline{w}_2^1 = (\bar{x}, 1, \bar{x}, 1).$$

Finally

$$\underline{w}'_1 = (\bar{x}, 0)$$

$$\underline{w}'^T = (\bar{x}, 0)$$

and

$$\underline{w}'_2 = (\bar{x}, 1)$$

$$\underline{w}'^F = (x, 0).$$

It would be possible to obtain the same result by using the following values

$$\underline{x}^T = (x, x) \quad \underline{y}_1^T = (0, 1) \quad \underline{u}^T = (0, 0)$$

$$\underline{x}^F = (\bar{x}, \bar{x}) \quad \underline{y}_1^F = (1, 0) \quad \underline{u}^F = (0, 0)$$

and computing

$$\bar{x}\bar{y}_1 + y_1u.$$
 ■

Contrary to the other procedures, Procedure 6 is not obvious. Therefore, it is necessary to justify it.

Assume we have a function

$$f(y_1, \dots, y_i, \dots, y_n, \underline{x}) = a + by_i + c\bar{y}_i \quad \text{where } bc = 0.$$

Let

$$f_1 = f(y_1, \dots, 0, \dots, y_n, \underline{x})$$

$$f_2 = f(y_1, \dots, 1, \dots, y_n, \underline{x})$$

and

$$m = f_1 \cdot f_2$$

$$M = f_1 + f_2.$$

Theorem 8: We have

$$f(y_1, y_2, \dots, u, \dots, y_n, \underline{x}) = m + Mu.$$

Proof: On one hand we have

$$f(y_1, y_2, \dots, u, \dots, y_n, \underline{x}) = a + bu + cu = a + (b+c)u$$

and on the other hand

$$m + Mu = (a+b)(a+c) + (a+b+c)u$$

$$= a + bc + au + (b+c)u$$

$$= a + (b+c)u.$$

If we use the T and F vectors and Theorem 8, we obtain

$$\underline{w}^T = \underline{m}^T + \underline{M}^T \cdot \underline{u}^T = \underline{m}^T = \underline{w}_1$$

and

$$\underline{w}^F = \underline{m}^F + \underline{M}^F \cdot \underline{u}^F = \underline{m}^F \cdot \underline{M}^F = \underline{\bar{m}}^T \cdot \underline{\bar{M}}^T$$

$$= (\underline{w}_1^0 \cdot \underline{w}_1^1) (\underline{w}_1^0 + \underline{w}_1^1).$$

Finally if we reduce the dimension of \underline{p}_1 and \underline{p}_2 , the procedure is justified. ■

In summary, we see that by using these vector operations over the characteristic vector \underline{p} , numerous classical operations requiring the manipulation of large Boolean expressions can now be carried out using simple operations (algorithms) over the components of these vectors.

REFERENCES

1. F. F. Sellers, M. Y. Hsiao, and L. W. Bearnson, "Analyzing Errors with the Boolean Difference," IEEE Trans. Computers, vol. C-17, pp. 676-683, July 1968.
2. S. S. Yau and Y. S. Tang, "Generation of Shortest Test Sequences for Individual Faults of Sequential Circuits," unpublished.
3. Y. Levendel, "Algebraic Test and Synchronizing Sequence Generation for Digital Switching Circuits," Ph. D. Dissertation, Electrical Engineering Department, University of Southern California, Los Angeles, 1977.
4. S. G. Chappell, "LAMP: Automatic Test Generation for Asynchronous Digital Circuits," The Bell Telephone Technical Journal, vol. 5, pp. 1477-1503, October 1974.
5. S. C. Lee, "Vector Boolean Algebra and Calculus," IEEE Trans. Computers, vol. C-25, pp. 865-874, September 1976.

DECOMPOSITION OF MULTIPLE-VALUED LOGIC FUNCTIONS

Jürgen Fricke

Ruhr-Universität Bochum
Federal Republic of Germany

1. Abstract

This paper deals with the problem of designing multiple-valued combinational functions and memory functions by complex modules. It is shown how logical functions can be described by Postian vectors or by polynomials which are based on operations of the ordinary arithmetic such as addition and multiplication. Two equations $u = S v$ and $u' = S v'$ are derived, by which all problems arising with two-level decomposition are solved. The solution to these problems is demonstrated by two examples.

2. Introduction

This paper deals with the problem of realizing multiple-valued combinational functions and multiple-valued memory functions by complex modules. As Wood¹ points out the design of multiple-valued logic circuits is based on a composition algebra. A composition algebra is any algebraic system (R, S, o) with relations R and operations o defined on set S such that S is closed under every operation o . The closure property of multiple-valued composition algebras allows arbitrary interconnection (or composition) of modules which realize the operations of the algebra. That is, the output state value of any module can be used as an input state value for any other gate. The mathematical term "operation" corresponds to the technical term "fundamental module", i.e. the algebraic description of a multiple-valued logical function $f(x_1, \dots, x_n)$ specifies which fundamental

modules are used for the realization of this function and how they are to be interconnected. For an economical realization normally the algebraic expression has to be minimized. The algebras of Allen, Givone² and Vranesic et al.³ were developed for this reason.

A complex module realizes a logical function which is more complicated than the function of a fundamental module. An algebraic expression of the function $f(x_1, \dots, x_n)$ does not show how to interconnect any complexⁿ module for the realization of this function. This is why in this paper it is not necessary to make use of a special composition algebra. All logical functions are represented by polynomials based on the ordinary arithmetic. As far as it is known to the author, this method for

the description of multiple-valued logical functions was first introduced by Dujmović⁴ in 1969.

3. Description of multiple-valued combinational functions and memory functions

The input state vector $\delta(i) := [\delta_j(i)]$, $j = 1, \dots, n$, is a vector whose components are coefficients of the n -digit base Θ -representation of the non-negative integers

$$i = \sum_{k=0}^{n-1} \alpha_k(i) \Theta^k, \quad \Theta \geq 2, \quad \alpha_k = 0, 1, \dots, \Theta-1, \quad (1)$$

with $\delta_j(i) = \alpha_{j-1}(i)$. In the product-term Θ^k of equ. (1) k is an exponent of Θ . The vector $\delta(i)$ is defined for $n \geq \log_{\Theta}(i-1)$. For example, with $\Theta=2$, $n=4$ is $\delta(5) = [1010]^T$, with $\Theta=3$, $n=4$ is $\delta(5) = [2100]^T$. By means of $\delta(i)$ the vector $\underline{k} := [k_i]$, $i=0, 1, \dots, \Theta^n-1$, is defined as a vector whose components are products of the Θ -valued variables x_1, \dots, x_n . The i^{th} component of vector \underline{k} is

$$k_i := \prod_{\mu=1}^n x_{\mu}^{\delta_{\mu}(i)} \quad (2)$$

In this equation is $\delta_{\mu}(i)$ the exponent of x_{μ} . Example: With the variables x_1, x_2 and $\Theta=3$, one obtains

$$\underline{k} = [1 \ x_1 \ x_1^2 \ x_2 \ x_1 x_2 \ x_1^2 x_2 \ x_2^2 \ x_1 x_2^2 \ x_1^2 x_2^2].$$

A Θ -valued combinational function $z=f(x_1, \dots, x_n)$

is described by the vector $\underline{u} := [u_i]$, $i=0, 1, \dots, \Theta^n-1$. The components u_i of this vector are

elements of the set $\{0, 1, \dots, \Theta-1\}$. They are values of the function corresponding to the input state vector $\delta(i)$. Denoting the present state value of a memory function by y , the next state value by y' and the next state value which is generated by the input state of vector $\delta(i)$ with u_i' , a memory

*) Underlined lower case letters denote column-vectors, row-vectors are described by transpose T . Matrices are denoted by underlined upper case letters.

function $y' = f(x_1, \dots, x_n, y)$ is described by the vector $\underline{u}' = [u'_i]$, $i=0, 1, \dots, \Theta^n - 1$. The components of this vector are elements of the set $\{0, 1, \dots, \Theta - 1, y\}$.

According to Dujmović a Θ -valued logical function can be described by means of the operations addition and multiplication:

$$z = f(x_1, \dots, x_n) = a_0 k_0 + a_1 k_1 + \dots + a_{\Theta^n - 1} k_{\Theta^n - 1} = \underline{a}^T \underline{k} \quad (3)$$

$$y' = f(x_1, \dots, x_n, y) = a'_0 k_0 + a'_1 k_1 + \dots + a'_{\Theta^n - 1} k_{\Theta^n - 1} = \underline{a}'^T \underline{k}.$$

The intrinsic properties of the functions $z = f(x_1, \dots, x_n)$ and $y' = f(x_1, \dots, x_n, y)$ in these expressions are contained in the vectors $\underline{a} = [a_i]$ and $\underline{a}' = [a'_i]$, $i=0, 1, \dots, \Theta^n - 1$, respectively. In ⁵ it is shown, that the vectors \underline{a} and \underline{a}' can be transformed into the vectors \underline{u} and \underline{u}' , respectively by means of a matrix $\underline{M}(\Theta, n)$:

$$\underline{u} = \underline{M}(\Theta, n) \underline{a} \quad (4)$$

$$\underline{u}' = \underline{M}(\Theta, n) \underline{a}'.$$

For $n=1$ the matrix $\underline{M}(\Theta, n) = [m_{ij}(\Theta, 1)]$, $i, j=0, 1, \dots, \Theta - 1$, is described by $m_{ij}(\Theta, 1) = i^j$. For arbitrary Θ and n , the Matrix $\underline{M}(\Theta, n)$ can be derived recursively from the matrix $\underline{M}(\Theta, n-1)$:

$$\underline{M}(\Theta, n) = \underline{M}(\Theta, 1) \times \underline{M}(\Theta, n-1), \quad (5)$$

where the operator " \times " is the Kronecker-product of two matrices.

The matrix $\underline{M}(\Theta, 1)$ is non-singular, hence there exists an inverse $\underline{M}^{-1}(\Theta, 1)$. By application of the rules for the inverse of a Kronecker-product the inverse of $\underline{M}(\Theta, n)$ can be recursively derived from the inverse of $\underline{M}(\Theta, n-1)$:

$$\underline{M}^{-1}(\Theta, n) = (\underline{M}(\Theta, 1) \times \underline{M}(\Theta, n-1))^{-1} \quad (6)$$

$$= \underline{M}^{-1}(\Theta, 1) \times \underline{M}^{-1}(\Theta, n-1).$$

The matrix $\underline{M}^{-1}(\Theta, n)$ transforms the vectors \underline{u} and \underline{u}' into the vectors \underline{a} and \underline{a}' , respectively. From now on, all vectors which describe a logical function according to the vectors \underline{a} and \underline{a}' are considered as being transformed. All vectors which describe a logical function according to the vectors \underline{u} and \underline{u}' are considered as being untransformed. They are denoted as Postian vectors, because their components are elements of the sets $\{0, 1, \dots, \Theta - 1\}$ and $\{0, 1, \dots, \Theta - 1, y\}$, respectively. For convenience, the arguments Θ and n of the matrices $\underline{M}(\Theta, n)$ and $\underline{M}^{-1}(\Theta, n)$ are omitted.

4. Description of the circuit structure

A block diagram of multiple decomposition of a combinational function or a memory function is given by fig.1. In the case of combinational functions the function $z = f(x_1, \dots, x_n)$ is decomposed into one function

$$z = g(r_1, \dots, r_m) = \underline{b}^T \underline{t} \quad (7)$$

and m functions

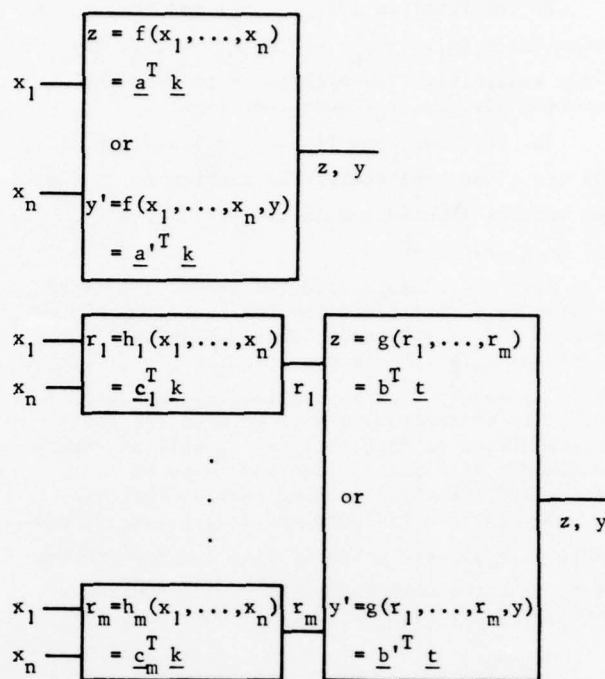


Fig.1. Block diagram of multiple decomposition

$$r_v = h_v(x_1, \dots, x_n) = \underline{c}_v^T \underline{k}, \quad v=1, \dots, m. \quad (8)$$

In the case of memory functions the function $y' = f(x_1, \dots, x_n, y)$ is decomposed into one function

$$y' = g(r_1, \dots, r_m, y) = \underline{b}'^T \underline{t} \quad (9)$$

and m functions

$$r_v = h_v(x_1, \dots, x_n) = \underline{c}_v^T \underline{k}, \quad v=1, \dots, m. \quad (10)$$

In these equations there are $\underline{b} = [b_i]$ and $\underline{b}' = [b'_i]$, $i=0, 1, \dots, \Theta^m - 1$, the transformed vectors of the functions $z = g(r_1, \dots, r_m)$ and $y' = g(r_1, \dots, r_m, y)$, respectively. The vectors $\underline{c}_v = [c_{iv}]$, with $i=0, 1, \dots, \Theta^n - 1$ and $v=1, \dots, m$, are the transformed vectors of the functions h_1, \dots, h_m . The corresponding Postian vectors are $\underline{v} = \underline{M} \underline{b}$, $\underline{v}' = \underline{M} \underline{b}'$ and $\underline{w}_v = \underline{M} \underline{c}_v$. The vector $\underline{t} = [t_j]$, $j=0, 1, \dots, \Theta^m - 1$, in equ.(7) is defined by

$$t_j := \prod_{v=1}^m \delta_{v-1}(j), \quad j=0, 1, \dots, \Theta^m - 1. \quad (11)$$

Ledley and Huang ⁶ have stated three typical problems for the design of multiple-valued combinational circuits with a structure as shown in fig.1:

1. The function $g(r_1, \dots, r_m)$ and the functions $h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n)$ are given. The problem is to find the unknown function $f(x_1, \dots, x_n, y)$.

2. The function $f(x_1, \dots, x_n)$ and the functions $h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n)$ are given explicitly. The problem is to find the function $g(r_1, \dots, r_m)$ such that $f=g$.

3a. The functions $f(x_1, \dots, x_n)$ and $g(r_1, \dots, r_m)$ are given explicitly. The problem is to find the unknown functions $h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n)$ such that $f=g$.

This paper deals with the second and third problem only. The first problem is a composition problem and is solved by substituting the functions h_1, \dots, h_m in the function $g(r_1, \dots, r_m)$, which is easily done by application of equ. (21) of ch.5. The third problem will be used for the decomposition of combinational as well as memory functions. If a memory function is to be decomposed the third problem runs as follows:

3b. The memory functions $f(x_1, \dots, x_n, y)$ and $g(r_1, \dots, r_m, y)$ are given explicitly. The problem is to find the unknown combinational functions $h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n)$ such that $f=g$.

5. Derivation of equations for the solution

In this section two equations $\underline{u} = \underline{S} \underline{v}$ and $\underline{u}' = \underline{S} \underline{v}'$ are derived, in which the Postian vectors \underline{v} and \underline{v}' of function g are transformed into the Postian vectors \underline{u} and \underline{u}' , respectively, of function f . The matrix \underline{S} describes the functions h_1, \dots, h_m . The derivation of this matrix is the same for both combinational functions and memory functions. Therefore it is carried out for combinational functions only.

Equ. (11) is substituted in equ. (7). With $f=g$, writing the resulting equation in sum of products form, it follows:

$$f(x_1, \dots, x_n) = \sum_{j=0}^{\Theta^m-1} b_j \prod_{v=1}^m r_v^{\delta_{v-1}(j)}. \quad (12)$$

With equ. (10) $r_v = h_v(x_1, \dots, x_n) = \underline{c}_v^T \underline{k}$ is

$$f(x_1, \dots, x_n) = \sum_{j=0}^{\Theta^m-1} b_j \prod_{v=1}^m (\underline{c}_v^T \underline{k})^{\delta_{v-1}(j)}. \quad (13)$$

In the last expression only the product $\underline{c}_v^T \underline{k}$ is dependent on the variables x_1, \dots, x_n . The function $r_v = h_v(x_1, \dots, x_n) = \underline{c}_v^T \underline{k}$ is described by the Postian vector \underline{w}_v . Substituting the components of the input state vector $\underline{\delta}(i)$ in the variables x_1, \dots, x_n according to the equation $x_\mu = \delta_{\mu-1}(i)$, for $i=0, 1, \dots, \Theta^n-1$ the vector $\underline{w}_v := [\underline{w}_{iv}]$ is obtained. From equ. (13) follows by the same substitution:

$$u_0 = \sum_{j=0}^{\Theta^m-1} b_j \prod_{v=1}^m \delta_{v-1}(j) w_{0v}$$

$$\begin{aligned} u_1 &= \sum_{j=0}^{\Theta^m-1} b_j \prod_{v=1}^m \delta_{v-1}(j) w_{1v} \\ &\dots \dots \dots \end{aligned} \quad (14)$$

$$u_{\Theta^n-1} = \sum_{j=0}^{\Theta^m-1} b_j \prod_{v=1}^m \delta_{v-1}(j) w_{\Theta^n-1, v}$$

With

$$p_{ij} := \prod_{v=1}^m w_{iv}^{\delta_{v-1}(j)} \quad (15)$$

and

$$\underline{P} := [p_{ij}] \quad \begin{matrix} i=0, 1, \dots, \Theta^n-1 \\ j=0, 1, \dots, \Theta^m-1 \end{matrix} \quad (16)$$

follows

$$\underline{u} = \underline{P} \underline{b}. \quad (17)$$

As shown by equ. (15), for $j = \Theta^{v-1}$ and $v=1, \dots, m$ the column j of the matrix \underline{P} is equal to the vector \underline{w}_v .

For memory functions

$$\underline{u}' = \underline{P} \underline{b}' \quad (18)$$

is obtained in the same way.

The equation $\underline{u} = \underline{P} \underline{b}$ transforms the vector \underline{b} of function g into the vector \underline{u} of function f . The right-hand side of $\underline{u} = \underline{P} \underline{b}$ is multiplied by the unity matrix $\underline{I} = \underline{M}^{-1} \underline{M}$ which yields

$$\underline{u} = (\underline{P} \underline{M}^{-1}) (\underline{M} \underline{b}). \quad (19)$$

The factor $\underline{M} \underline{b}$ of this expression is already known as the Postian vector $\underline{v} = \underline{M} \underline{b}$, the factor

$$\underline{S} := \underline{P} \underline{M}^{-1} \quad (20)$$

ought to be examined more closely.

It can be shown that for the elements s_{ik} of matrix \underline{S} ,

$$s_{ik} \in \{0, 1\}$$

holds true. There is exactly one unity element in each row of matrix \underline{S} . The reader may verify this. A proof is given in ⁵. A second property can be shown considering fig. 1. By means of the functions h_1, \dots, h_m an output state of the variables r_1, \dots, r_m is assigned to each input state of the variables x_1, \dots, x_n . The input states of the variables x_1, \dots, x_n identify the rows of matrix \underline{S} , the output states of the variables r_1, \dots, r_m identify the columns. The element s_{ik} is equal to 1, provided that the input state i of the variables x_1, \dots, x_n corresponds with the output state k of the variables r_1, \dots, r_m . The states i and k are defined by the state vectors $\underline{\delta}(i)$ and $\underline{\delta}(k)$, respectively. Otherwise s_{ik} is zero.

From eqs. (19) and (20) follows

$$\underline{u} = \underline{S} \underline{v} \quad (21)$$

and

$$\underline{u}' = \underline{S} \underline{v}'. \quad (22)$$

With the property of the matrix \underline{S} (one unity-element per row) and the last two equations an important condition for the decomposition of multiple-valued functions is derived:

$$\text{if } s_{ik} = 1 \begin{cases} \text{then } u_i = v_k \\ \text{or } u_i' = v_k', \text{ respectively} \\ \text{for } i = 0, 1, \dots, \Theta^n - 1, \\ \text{and } k = 0, 1, \dots, \Theta^m - 1. \end{cases} \quad (23)$$

This condition is helpful in solving the problems 2 and 3.

6. Solution

At first the solution of problem 2 is demonstrated by means of an example. The 3-valued combinational function $f(x_1, x_2)$ is given by the Postian vector $\underline{u} = [210 \ 210 \ 000]^T$. The functions $h_1(x_1, x_2)$ and $h_2(x_1, x_2)$ are given by the Postian vectors $\underline{w}_1 = [222 \ 222 \ 000]^T$ and $\underline{w}_2 = [210 \ 210 \ 210]^T$, respectively. The problem is to find the unknown function $g(r_1, r_2)$. After constructing the matrix \underline{S} , with $\underline{S} \underline{v} = \underline{u}$ one arrives at:

$$\begin{array}{c|cccccccc} & r_1 r_2 & 00 & 01 & 10 & 11 & 20 & 21 & 02 & 12 & 22 \\ \hline x_1 x_2 & 00 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ & 20 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 01 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ & 11 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ & 21 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 02 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ & 12 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ & 22 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 2 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In this example there is $s_{08} = 1$ and $u_0 = 2$. Therefore, condition (23) yields $v_8 = 2$. The complete solution for the function $g(r_1, r_2)$ is given by $\underline{v} = [0-0 \ 0-1 \ 0-2]^T$. Columns 1, 4 and 7 of the matrix \underline{S} do not contain any 1-element. For this reason the elements v_1 , v_4 and v_7 of the vector \underline{v} are "don't care"-elements "-". On the other hand the two elements s_{08} and s_{38} in column 8 of the matrix \underline{S} are equal to "1". From this follows, that there is no solution to the problem unless $u_0 = u_3$.

Finally, the solution of a problem of type 3 is demonstrated by means of a 3-valued memory function. The function $f(x_1, x_2, y)$ is given by the Postian vector $\underline{u}' = [yy0 \ yy1 \ 012]^T$, the function $g(r_1, r_2, y)$ is given by the Postian vector $\underline{v}' = [yy0 \ yy1 \ yy2]^T$. The problem is to find the unknown functions $h_1(x_1, x_2)$ and $h_2(x_1, x_2)$.

Considering the condition (23), the equation $\underline{S} \underline{v}' = \underline{u}'$ is written as

$$\begin{bmatrix} s_{00} & s_{01} & 0 & s_{03} & s_{04} & 0 & s_{06} & s_{07} & 0 \\ s_{10} & s_{11} & 0 & s_{13} & s_{14} & 0 & s_{16} & s_{17} & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ s_{30} & s_{31} & 0 & s_{33} & s_{34} & 0 & s_{36} & s_{37} & 0 \\ s_{40} & s_{41} & 0 & s_{43} & s_{44} & 0 & s_{46} & s_{47} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ y \\ 0 \\ y \\ y \\ 1 \\ y \\ y \\ 2 \end{bmatrix} = \begin{bmatrix} y \\ y \\ 0 \\ y \\ y \\ 1 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

All elements of the matrix \underline{S} , which are denoted by s_{ik} , are undetermined and can be 0 or 1. The

matrix \underline{P} , which explicitly contains the solution for the function $h_1(x_1, x_2)$ and $h_2(x_1, x_2)$ in form of the vectors \underline{w}_1 and \underline{w}_2 , is derived by multiplying equ.(20) on both sides by \underline{M} , i.e.

$$\underline{P} = \underline{S} \underline{M}. \quad (24)$$

As already mentioned with regard to equ.(15) the vectors \underline{w}_1 and \underline{w}_2 are equal to the columns 1 and 3, respectively, of the matrix \underline{P} . Consequently these vectors are derived by multiplying the matrix \underline{S} by columns 1 and 3 of the matrix $\underline{M}(3,2)$:

$$\underline{w}_1 = \begin{bmatrix} s_{01} + s_{04} + s_{07} \\ s_{11} + s_{14} + s_{17} \\ 2 \\ s_{31} + s_{34} + s_{37} \\ s_{41} + s_{44} + s_{47} \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}, \quad \underline{w}_2 = \begin{bmatrix} s_{03} + s_{04} + 2s_{06} + 2s_{07} \\ s_{13} + s_{14} + 2s_{16} + 2s_{17} \\ 0 \\ s_{33} + s_{34} + 2s_{36} + 2s_{37} \\ s_{43} + s_{44} + 2s_{46} + 2s_{47} \\ 1 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

Denoting the number of values ℓ , which appear as a component of \underline{u} or \underline{u}' with μ_ℓ and the number of values ℓ which appear as a component of \underline{v} or \underline{v}' with ν_ℓ , where $\ell \in \{0, 1, \dots, \Theta - 1, y\}$, the total number of solutions of the matrix \underline{S} is obtained from the product

$$\kappa = \prod_{\ell=0}^{\Theta-1} \nu_\ell^{\mu_\ell} \cdot \mu_y \quad (25)$$

There are $\kappa = 1^2 1^1 6^4 = 1296$ solutions for the matrix \underline{S} of the running example. One solution is obtained by setting $s_{00} = s_{31} = s_{44} = 1$. From this follows

$$\underline{w}_1 = [0 \ 1 \ 2 \ 1 \ 1 \ 2 \ 2 \ 2 \ 2]^T, \\ \underline{w}_2 = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 2]^T.$$

i.e.

$$h_1(x_1, x_2) = \max(x_1, x_2), \\ h_2(x_1, x_2) = \min(x_1, x_2).$$

This example is showing that there are only

solutions to a problem of type 3 if each component of the vectors \underline{u} , \underline{u}' appears at least once as a component of the vectors \underline{v} , \underline{v}' , respectively. It is also showing that only such multiple-valued memory functions can be treated which can be described by Postian vectors \underline{u}' or \underline{v}' , as defined in section 2.

7. Conclusions

It has been shown how the design of multiple-valued combinational functions and a special class of memory functions is treated by operations of the ordinary arithmetic such as addition and multiplication. All problems arising with two-level decomposition are solved by the equations $\underline{u} = \underline{S} \underline{v}$ and $\underline{u}' = \underline{S} \underline{v}'$, respectively, where \underline{u} , \underline{v} , \underline{u}' and \underline{v}' are Postian vectors and \underline{S} is a matrix having only one unity-element per row.

8. References

- 1 P.E. Wood: "Switching theory", New York: McGraw-Hill, 1968.
- 2 C.M. Allen, D.D. Givone: "A minimization technique for multiple-valued logic systems", IEEE Trans. Comp., C-17, 1968, pp. 182-184.
- 3 Z.G. Vranesic, E.S. Lee, K.C. Smith: "A many-valued algebra for switching systems", IEEE Trans. Comp., C-19, 1970, pp.964-971.
- 4 J.J. Dujmović: "The interpolated algebraic equivalence of functions in k-valued logic", 5th Yugoslav Internat. Symp. for Information Processing, Bled, Oct. 8-11, 1969, paper A 1-5.
- 5 J. Fricke: "Dekomposition mehrwertiger logischer Funktionen - Ein Beitrag zur Realisierung mehrwertiger digitaler Schaltungen", Schriftenreihe des Lehrstuhls für Meß- und Regelungstechnik, Ruhr-Universität, 4630 Bochum, West-Germany. Dissertation 1977
- 6 R.S. Ledley, H.K. Huang: "Multy-valued logic design and Postian matrices", Proc. Internat. Symp. on Multiple-valued Logic, May 13-16, 1975, Indiana-University, Bloomington, pp. 67-75.

THEORY AND DESIGN OF MULTIVALUED MEMORY ELEMENTS

J.L.HUERTAS, J.I.ACHA, G.SANCHEZ GOMEZ

Departamento de Electricidad y Electrónica
Facultad de Ciencias
Universidad de Sevilla

Abstract

An extension of the concept of Boolean memories is presented. A new formulation is given which allows to investigate multivalued memory elements. These elements define a class of asynchronous memory circuits, particular cases of which are the usual multistables. A systematic method of synthesis is developed. The method is based on a simple interpretation of the stability constraints which are imposed by an state matrix.

1. Introduction

The synthesis of multivalued sequential networks has attracted the attention of a number of researchers during the last years. Different approaches to the problem of designing multivalued (MV) memory elements have been reported ¹⁻⁵. However, all of these approaches have been directed toward the synthesis of p-valued elements with p stable states. They are called multivalued flip-flop. Examples of both synchronous and asynchronous multistables which belong to that class may be found elsewhere ¹⁻⁵. For any base (including binary) asynchronous flip-flops are interesting by themselves and because they are the basic building block for the synthesis of synchronous multistables. The formers are realized by the closed interconnection of logic operators which possess the Involution Property.

Danielsson ⁶ pointed out for binary flip-flops that they are comprised in a more general class of memory elements which he called Boolean memories. The distinctive characteristic of these elements is to exhibit m stable states for a p-valued logic. Although they have not been extensively used, exam-

ples of applications can be found ⁷. Principally, they are interesting when the information is not coded in a pure binary way.

The purpose of this paper is the generalization of the concepts developed in ⁶ to the multivalued case. It has a theoretical interest since appears to be a systematic treatment to the synthesis of asynchronous memory elements for any base. From a practical point of view, a synthesis method is given which applies for any base and any number of stable states. It may be of value when the information to be stored is not coded in a conventional way.

II. Preliminary concepts

This section is devoted to the definition of some basic concepts. Some of them were introduced in ⁶ but are generalized in this paper.

Definition 1. Memory machine: A memory machine (MM) of order m is an asynchronous sequential machine characterized by a set of internal states $\{\sigma_1, \sigma_2, \dots, \sigma_m\}$ and a set of inputs $\{I_0, I_1, I_2, \dots, I_m\}$ and having the transition table given in Table 1.

Definition 2. p-valued memory element: A p-valued memory element (PVME) is a circuit realization of a memory machine. Any input symbol I_i is represented by a combination of the p-valued input variables X_1, X_2, \dots, X_p and any state σ_j is coded into a combination of the p-valued state variables A_1, A_2, \dots, A_n .

Definition 3. p-valued flip-flop: A p-valued flip-flop (PVFF) is a p-valued memory element of order p.

It is important to note that Definition 1 does not impose any restriction to the existence of other columns that the ones of the set $\{l_0, l_1, \dots, l_m\}$. With that remark it should be clear that Definition 3 applies for any multistable.

We can see from Table 1 that no transitions occur for the input symbol l_0 . We will say that the memory element is in its essential memory mode. Whenever the input symbol is $l_i \neq l_0$, we will say that the memory element is in the setting mode.

As Danielsson pointed out in ⁶, memory elements have an important property. In the setting mode, the next state is dependent only on the input variables X_1, X_2, \dots, X_m . It means that the circuit behaves like a combinational circuit. The feedback action only takes place when the circuit is in the essential memory mode can be reduced to the analysis of the equilibrium states of an input-free feedback network. It allows the development of a synthesis method rather unconventional but very adequate for this kind of sequential machines.

III. General structure

We repeat here a well-known result due to Unger ⁸. He has shown that any closed feedback loop which can take two or more equilibrium states must contain at least one amplifier. As a generalization of a condition given in ⁶ we will state:

Assumption 1: Each amplifier is a logical operator with the Involution Property and viceversa.

Corollary 1: Each feedback loop contains at least an involutive operator (I-Operator).

In accordance with the above statement we can represent the general structure of a memory element in the form of Figure 1. We will consider the I-operator outputs as the feedback signals. Of course, the arrangement does not lead to a minimum number of feedback loops.

The block labelled Givone's operators is a combinational block which contains only some unitary gates which perform a logical transformation on the feedback signals. The transformation is defined by

9:

$$i_{X^j} = \begin{cases} p-1 & \text{iff } i \leq X \leq j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $p-1$ is the highest value of the algebra.

The other combinational block only contains AND and OR gates which are defined as:

$$\begin{aligned} \text{AND: } A \cdot B &= \text{MIN}(A, B) \\ \text{OR: } A + B &= \text{MAX}(A, B) \end{aligned} \quad (2)$$

A question arises when we consider the diversity of I-Operators which can be defined. We have been concerned with:

$$\begin{aligned} \text{Cycle: } A \xrightarrow{B} &= (A+B) \text{mod. } p \\ \text{Complement: } \bar{A} &= (p-1)-A \\ \text{Identity: } A &= A \end{aligned} \quad (3)$$

where A and B are logical variables which take values of the set $\{0, 1, \dots, p-1\}$ of a p -valued algebra.

We can distinguish two different classes of memory elements.

Definition 4. Pure p -valued memory element: Is a PVME with identical I-Operators in all the feedback loops.

Definition 5. Hybrid p -valued memory element: Is a PVME with different types of I-Operators.

IV. Binary memory elements

Danielsson has studied the p -valued memory elements for $p = 2$. He called boolean memories to this particular elements. In order to understand our general treatment, it is necessary a quick glimpse to the algebraic formulation which was introduced in ⁶.

For binary components, the I-Operators should be inverters and the combinational blocks of Figure 1 are reduced to one which only contains AND and OR gates. The behaviour of the system is represented by the set of equations:

$$\begin{aligned}
\bar{A}_1 &= f_1(A_2, \dots, A_n) \\
\bar{A}_2 &= f_2(A_1, A_3, \dots, A_n) \\
&\vdots \\
\bar{A}_n &= f_n(A_1, A_2, \dots, A_{n-1})
\end{aligned} \quad (4)$$

The synthesis method developed in ⁶ is based on an efficient use of the state matrix. The columns of the state matrix correspond to the different stable states of the memory element. The column associated with the state σ_i is formed by the code assigned to that state. The dimension of the state matrix is $n \times m$, where n is the number of feedback loops and m the number of the stable states.

The design procedure consists in expressing \bar{A}_i as a function of an auxiliary variable P_j as follows:

$$[\bar{A}] = [\sigma] [P] \quad (5)$$

where

$$\begin{aligned}
[\bar{A}]^T &= [\bar{A}_1, \bar{A}_2, \dots, \bar{A}_n] \\
[P]^T &= [P_1, P_2, \dots, P_n]
\end{aligned}$$

and $[\sigma]$ is the state matrix.

The auxiliary variables are determined by using:

$$[\bar{P}] = [\Sigma]^T [\bar{A}] \quad (6)$$

where $[\Sigma]$ is a matrix whose elements are the complement of the elements of $[\sigma]$.

As a result, the variable \bar{A}_i can be expressed by:

$$\bar{A}_i = \sum_{j=1}^m S_{ij} P_j \quad (7)$$

It should be clear that we use the elements of $[\sigma]$ which are logic 1's in equation (5) and the elements which are logic 0's in equation (6). An alternative expression for (5) is:

$$[A^*] = [\sigma] [P^1] \quad (8)$$

where

$$\begin{aligned}
[P^1]^T &= [P_1^1, P_2^1, \dots, P_m^1] \\
[A^*]^T &= [A_1^*, A_2^*, \dots, A_n^*]
\end{aligned}$$

We use A_i^* to represent the result of the applica

tion of an I-operator on A_i .

Expression (5) is the base for our generalization to non-binary algebras as we will see in the next section.

A parallel development leads to the set of equations:

$$[A] = [\Sigma] [\bar{S}] \quad (9)$$

where S is obtained from:

$$[S] = [\sigma]^T [A] \quad (10)$$

V. Stability conditions for PVME

In this section we consider an input-free PVME in order to determine the equilibrium constraints which are valid in its essential memory mode. From the Figure 1 any PVME has to satisfy the following set of logic equations:

$$\begin{aligned}
A_1^* &= f_{1k}(A_2^k, A_3^k, \dots, A_n^k) \\
A_2^* &= f_{2k}(A_1^k, A_3^k, \dots, A_n^k) \\
&\vdots \\
A_n^* &= f_{nk}(A_1^k, A_2^k, \dots, A_{n-1}^k)
\end{aligned} \quad (11)$$

where the functions f_{ik} contain only the MAX and MIN operators. As it can be seen from (11) the Givone's operators used in the first stage of the combinational block are of the same order. Furthermore, the following theorem gives more information on the functions f_{ij} .

Theorem 1: The function f_{ij} does not depend on the operand A_i^j except for taking the value j^* .

Proof: Let us assume that A_i^j contributes to f_{ij} . Then, for one or more combination of the remaining operands, A_i^* depends directly on A_i^j . That dependence can be expressed as $f_{ij} = r \cdot A_i^j$. This equation represents a nonequilibrium condition except for $r = j^*$. But nonequilibrium conditions cannot exist in the essential memory mode. It concludes the proof.

Now, we will study the equilibrium conditions of the memory elements.

Definition 6: Let σ_i and σ_j be two columns of the transformed state matrix $[\Sigma]$. We will say that σ_i and σ_j contains a k-level stability pair if at least two rows form a submatrix of the following type:

$$\begin{array}{cc} \sigma_i & \sigma_j \\ \cdot & \cdot \\ \cdot & k \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & s \\ \cdot & k \\ \cdot & \cdot \end{array}$$

provided that $s, r \neq k$, although s and r can be equal.

Theorem 2. Stability theorem: Let $[\Sigma]$ represent a p-valued memory element. The variables A_i^* can be expanded as a function of the variables $k_{A_j^k}$, $\psi_j = i$ iff each pair of columns of $[\Sigma]$ have at least one k-level stability pair.

Alternatively, every pair of columns of $[\sigma]$ must have at least one k^* -level stability pair.

Proof: A_i^* can be expressed as a function of $k_{A_j^k}$ iff $k_{P_s^k}$ depends also of $k_{A_j^k}$. It is equivalent to say that the s-th column of $[\Sigma]$ contains at least one the value k .

Let to suppose that k^* does not appear at any column of $[\sigma]$. At this case, the value k is missing at any column of $[\Sigma]$, which means that A_i^* cannot be expanded as a function of $k_{A_j^k}$ in contradiction with the hypothesis.

Let σ_a and σ_b be two columns of $[\sigma]$. We will assume that in some row, σ_b has the value k^* and σ_a a different logical value r . Let to suppose that this row corresponds to A_1^* . Since:

$$A_1^* = f_{1k}(k_{A_1^k}, k_{A_2^k}, \dots, k_{A_{l+1}^k}, \dots, k_{A_n^k}) \quad (12)$$

at least one of the remaining signals, say $k_{A_h^k}$, must bring about the different values in f_{1k} . In fact, with this assumption we can write:

$$A_1^* = r k_{A_h^k} + k^* k_{A_1^k} \quad (13)$$

when $A_h = k$, as f_{1k} must follow $k_{A_h^k}$, $A_1^* = r$. It means $A_h^* = k^*$ for σ_a . When $A_h \neq k$, the value of A_1 depends on itself as the theorem 1 shows. In that case $A_1^* = k^*$ and $A_h = s \neq k^*$ for σ_b . It concludes the proof.

Corollary 2: If A_i^* can be expressed as a function of $k_{A_j^k}$ ($\psi_j \neq i$), neither the k's nor the r's ($r \neq k$) of a state matrix column can be covered by the corresponding configurations in any other state.

Proof: Let assume two columns σ_a and σ_b where the k's of σ_a are covered by the k's of σ_b . Then it is impossible to form a k-level stability pair.

As an example, we can consider the following matrix which does not satisfy the stability theorem.

$$\begin{array}{cccc} \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 \\ \begin{bmatrix} r_1 & r_2 & r_3 & k \\ k & r_4 & k & r_s \\ k & r_6 & r_7 & k \\ r_8 & k & r_9 & r_{10} \end{bmatrix} \end{array}$$

V. Synthesis procedure

Theorem 2 has shown the necessity of the existence of at least one k-level stability pair in the transformed state matrix. But that matrix, $[\Sigma]$, can have other stability pairs of different level. It is formalized by the following theorem.

Theorem 3: Let $[\sigma]$ be a state matrix whose transformed state matrix $[\Sigma]$ fulfills the Stability Theorem for $k = k_1, k_2, \dots, k_e$. For $[\sigma]$ we can always find one memory equation set for each k_i with the condition that the system is satisfied by and only by the given matrix columns.

Before the proof, we shall describe the general synthesis method which have been developed. In accordance with theorem 2, A_i^* can be expressed as a combination of $k_{A_1^k}$. For simplicity, we will introduce the auxiliary variables $k_{P_j^k}$ which allows to write A_i^* as a "linear" combination:

$$A_i^* = \sum_{j=1}^m \sigma_{ij} k_{P_j^k} \quad (14)$$

where Σ applies for the MAX operator and $k_{P_j^k}$ is a product of some of the signals $k_{A_1^k}$. The problem is now reduced to find $k_{P_j^k}$ for a given state matrix. An example can contribute to clarify the approach.

Let to design a pure 3-valued memory element with four feedback loops and four states and with the state matrix:

$$[\sigma] = \begin{bmatrix} 2 & 0 & 1 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix} \quad (15)$$

We will use a counterclockwise operator as the I-Operator. It is:

$$A_i^* = A_i^+ \quad (16)$$

Then, the transformed state matrix is:

$$[\Sigma] = \begin{bmatrix} 0 & 1 & 2 & 2 \\ 1 & 2 & 0 & 2 \\ 2 & 0 & 1 & 1 \\ 1 & 1 & 2 & 0 \end{bmatrix} \quad (17)$$

It can be observed that $[\Sigma]$ satisfies the stability theorem for $k = 0$. It allows to write expression (14) as:

$$\begin{aligned} A_1^+ &= 0 \cdot {}^0P_2^0 + 1 \cdot ({}^0P_3^0 + {}^0P_4^0) + 2 \cdot {}^0P_1^0 \\ A_2^+ &= 0 \cdot {}^0P_1^0 + 1 \cdot ({}^0P_2^0 + {}^0P_4^0) + 2 \cdot {}^0P_3^0 \\ A_3^+ &= 0 \cdot ({}^0P_3^0 + {}^0P_4^0) + 1 \cdot {}^0P_1^0 + 2 \cdot {}^0P_2^0 \\ A_4^+ &= 0 \cdot ({}^0P_1^0 + {}^0P_2^0) + 1 \cdot {}^0P_3^0 + 2 \cdot {}^0P_4^0 \end{aligned} \quad (18)$$

We must remember that any p-valued function can be expressed as:

$$f = \sum_{j=0}^p j \cdot f_j = \sum_{j=1}^p j \cdot f_j \quad (19)$$

where we have specified the value of the function in two alternative ways. The first procedure details the value of all of the cells which are different from zero. This is the normal method for writing a switching function. However, we have many representations which are theoretically equivalent but whose usefulness may depend on the circuit elements which are employed. In particular, we can write (14) without specifying the cells whose value is k^* . For the example:

$$A_1^+ = 0 \cdot {}^0P_2^0 + 1 \cdot ({}^0P_3^0 + {}^0P_4^0)$$

$$\begin{aligned} A_2^+ &= 0 \cdot {}^0P_1^0 + 1 \cdot ({}^0P_2^0 + {}^0P_4^0) \\ A_3^+ &= 0 \cdot ({}^0P_3^0 + {}^0P_4^0) + 1 \cdot {}^0P_1^0 \\ A_4^+ &= 0 \cdot ({}^0P_1^0 + {}^0P_2^0) + 1 \cdot {}^0P_3^0 \end{aligned} \quad (20)$$

Since the zeroes of $[\Sigma]$ mark the elements ${}^0A_i^0$ which are responsible of each ${}^0P_j^0$, we can determine them:

$$\begin{aligned} {}^0P_1^0 &= {}^0A_1^0 \\ {}^0P_2^0 &= {}^0A_3^0 \\ {}^0P_3^0 &= {}^0A_2^0 \\ {}^0P_4^0 &= {}^0A_4^0 \end{aligned} \quad (21)$$

We will call this type of expansion the canonical sum-of-products CS^{OP}.

In general, we can calculate k_p^k by mean of the equation system:

$$[k_p^k] = [k_\Sigma^k]^T [k_A^k] \quad (22)$$

which is the generalization of (6). A_i^* as a function of k_p^k is then obtained from:

$$[A^*] = [\sigma] [k_p^k] \quad (23)$$

It has to be noted that expression (22) is a binary equation which points out that k_p^k depends on k_A^k iff the element of row s and column j of $[\Sigma]$ is k .

Lemma 1: In a CS^{OP} one and only one of the products $i_p^i, i_p^i, \dots, i_p^i$ can be equal to the maximum value of the logic, $p-1$, at an equilibrium state.

Proof: First, let all products be zero. Then, the functions $f_{1i}, f_{2i}, \dots, f_{ni}$ are equal to i^* which in turn implies $iA_1^i = iA_2^i = \dots = iA_n^i = p-1$ which is in contradiction with the hypothesis.

Next, let to assume the particular expansion CS^{kP} and suppose that the product k_p^k , associated with to state σ_j , takes the value $p-1$. Only when the i -th row of $[\sigma]$ has one $r \neq k^*$, the product k_p^k appears explicitly in the function f_{ik} . Hence k_p^k holds the corresponding feedback signals at the value S ($S \neq k$). But since Corollary 2, every state

σ_i distinct of σ_j must have at least one k in any of the rows of $[\Sigma]$. Thus, any of the signals which are kept to S by $k_{p_j}^k$ is a factor in every product $k_{p_e}^k$ different from $k_{p_j}^k$. In that way, $k_{p_e}^k = 0$, $\forall e \neq j$. It concludes the proof.

We can construct an auxiliary matrix, $[M]$, whose columns are associated with the states of the PVME and whose rows are associated with the products $k_{p_i}^k$. The element m_{ij} is defined as:

$$m_{ij} = \begin{cases} p-1 & \text{iff the product } k_{p_i}^k \text{ takes} \\ & \text{the value } p-1 \text{ for the state } \sigma_j \\ 0 & \text{otherwise} \end{cases} \quad (24)$$

Taking into account the result stated by the Lemma 1, $[M]$ is a diagonal matrix:

$$[M] = \begin{bmatrix} p-1 & 0 & \dots & 0 \\ 0 & p-1 & \dots & 0 \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ 0 & 0 & \dots & p-1 \end{bmatrix} \quad (25)$$

Another conclusion which can be obtained from the Lemma 1 is the existence of an arbitrary partition on the set of the products $k_{p_j}^k$. We can divide that set into two disjoint subsets $R = \{R_1, R_2, \dots, R_c\}$ and $Q = \{Q_1, Q_2, \dots, Q_d\}$, which fulfills:

$$\sum_{i=1}^c R_i = \sum_{j=1}^d Q_j \quad (26)$$

where $c + d = m$ and the elements of R and Q are the binary variables $k_{p_i}^k$.

Expression (26) plays an important role since many times certain products or sums of products can be substituted by a simple feedback signal $k_{A_i}^k$. It allows a minimal hardware realization.

Proof of Theorem 3: Let to consider the expansion $\Sigma k_{p_j}^k$. Let $k_{p_j}^k$ be the product which is responsible of state σ_j . By writing $[M]$ it should be clear that a system of m logical equations can be formulated with at least m solutions.

Next, we must show that no extra solutions exist. If there is an extra solution, the dimension of $[M]$ should be $(m+1) \times (m+1)$. The extra solution

will be associated with a new state σ_{m+1} . Since:

$$k_{p_{m+1}}^k \neq f(k_{p_j}^k), \forall j \neq m+1 \quad (27)$$

the code for σ_{m+1} will be $A_i = 0, \forall i \in \{1, 2, \dots, n\}$ which is not a solution of the equation set. In conclusion, the new state σ_{m+1} cannot be an equilibrium state.

VI. Setting mode

The previous section have considered the equilibrium conditions for an input-free PVME. The input signals can be added to the input-free circuit in a direct way which is a straightforward extension of the method given in [6].

Since only one product variable can be $p-1$ for any equilibrium state, the simplest method for setting the memory to the state σ_a is forcing the value of $j_{p_a}^j$ to be $p-1$. An OR gate is enough to do that. The signal $j_{p_a}^j$ is then substituted by $X_a \cdot j_{p_a}^j$.

It is necessary to note that X_a can be binary or multivalued in our approach. When all the input signals are binary (logical values 0 and $p-1$), the only elements of the state matrix which can be reached are the ones shown in Table 1. However, when the input signals are multivalued, many other columns are also accesible. For convenience we can use bX_a^c in order to set the required states. It allows the use of multivalued input signals without the drawback of unwanted columns.

VII. Design example

In order to best illustrate the method explained above we will devote this section to the design of a 3-valued hybrid PVME 6 states and 5 feedback loops. The state matrix for that example is:

$$[\sigma] = \begin{bmatrix} 0 & 0 & 2 & 1 & 1 & 2 \\ 2 & 1 & 2 & 1 & 2 & 0 \\ 2 & 1 & 2 & 0 & 1 & 1 \\ 0 & 0 & 2 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 \end{bmatrix} \quad (28)$$

The starred operators have been chosen as: $A_1^+, A_2^+, \bar{A}_3, A_4^+, \bar{A}_5$.

Then the transformed state matrix is:

$$[\Sigma] = \begin{bmatrix} 1 & 1 & 0 & 2 & 2 & 0 \\ 1 & 0 & 1 & 0 & 1 & 2 \\ 0 & 1 & 0 & 2 & 1 & 1 \\ 2 & 2 & 1 & 0 & 0 & 1 \\ 2 & 2 & 2 & 1 & 2 & 0 \end{bmatrix} \quad (29)$$

which verifies the stability theorem for $k = 1$. Therefore, the product terms are:

$$\begin{aligned} {}^1P_1^1 &= {}^1A_1^1 {}^1A_2^1 \\ {}^1P_2^1 &= {}^1A_1^1 {}^1A_3^1 \\ {}^1P_3^1 &= {}^1A_2^1 {}^1A_4^1 \\ {}^1P_4^1 &= {}^1A_5^1 \\ {}^1P_5^1 &= {}^1A_2^1 {}^1A_3^1 \\ {}^1P_6^1 &= {}^1A_3^1 {}^1A_4^1 \end{aligned} \quad (30)$$

And the equation set:

$$\begin{aligned} A_1^+ &= 0({}^1P_1^1 + {}^1P_2^1) + 1({}^1P_4^1 + {}^1P_5^1) + 2({}^1P_3^1 + {}^1P_6^1) \\ A_2^+ &= 0({}^1P_6^1) + 1({}^1P_2^1 + {}^1P_4^1) + 2({}^1P_1^1 + {}^1P_3^1 + {}^1P_5^1) \\ \bar{A}_3 &= 0({}^1P_4^1) + 1({}^1P_2^1 + {}^1P_5^1 + {}^1P_6^1) + 2({}^1P_1^1 + {}^1P_3^1) \\ A_4^+ &= 0({}^1P_1^1 + {}^1P_2^1) + 1({}^1P_4^1 + {}^1P_5^1) + 2({}^1P_3^1 + {}^1P_6^1) \\ \bar{A}_5 &= 0({}^1P_1^1 + {}^1P_2^1 + {}^1P_3^1 + {}^1P_5^1) + 1({}^1P_4^1) + 2({}^1P_6^1) \end{aligned} \quad (31)$$

However, the hardware implementation may be simplified by considering the expression (19). The final equations are:

$$\begin{aligned} A_1^+ &= 1({}^1A_5^1 + {}^1A_2^1 {}^1A_3^1) + 2({}^1A_2^1 {}^1A_4^1 + {}^1A_3^1 {}^1A_4^1) \\ A_2^+ &= 1({}^1A_1^1 {}^1A_3^1 + {}^1A_5^1) + 2({}^1A_1^1 {}^1A_2^1 + {}^1A_2^1 {}^1A_4^1 + {}^1A_2^1 {}^1A_3^1) \\ \bar{A}_3 &= 1({}^1A_1^1 {}^1A_3^1 + {}^1A_2^1 {}^1A_3^1 + {}^1A_3^1 {}^1A_4^1) + 2({}^1A_1^1 {}^1A_2^1 + {}^1A_2^1 {}^1A_4^1) \\ A_4^+ &= 1({}^1A_5^1 + {}^1A_2^1 {}^1A_3^1) + 2({}^1A_2^1 {}^1A_4^1 + {}^1A_3^1 {}^1A_4^1) \\ \bar{A}_5 &= 1({}^1A_5^1) + 2({}^1A_3^1 {}^1A_4^1) \end{aligned} \quad (32)$$

Figure 2 shows an implementation of this equation set. It should be remarked that each one of

the signals A_2^* , A_3^* and A_5^* depend on itself for taking the value 1.

A different problem arises when we are interested in implementing a PVME with a given $[\Sigma]$, but with the freedom of selecting the I-Operators to be placed on the feedback loops. This problem allows to select the I-Operators in such a way that A_i^* does not depend on kA_i^k . In our example it can be done by selecting $A_i^* = A_i$ for $i \in \{1, 2, 3, 4, 5\}$.

VIII. Conclusions

Unger's theorem has been used to define a class of multivalued asynchronous sequential machines. The results have been obtained as an extension of the concept of Boolean memories ⁶.

We have given an alternative formulation to the binary case. The new formulation can be directly extended to multivalued systems. Conditions for stability have been studied and give more insight into the corresponding conditions developed in ⁶ for binary memories.

A synthesis procedure have been described which allows the design of any p-valued m-state memory element. It is based on the use of a prescribed state assignment which is given by the state matrix. The technique has been applied to some examples (two of them are given in the text). Practical circuits constructed with C-MOS integrated elements have shown a performance which agrees with the theoretical predictions.

References

- 1 WOJCIK, A.S.: "Multivalued Asynchronous sequential circuits". Proc. of the 1974 ISMVL, May 1974, pp. 165-166.
- 2 HIGUCHI, T. and KAMEYAMA, M.: "Static-hazard-free T-gate for ternary memory element". Proc. of the ISMVL, pp. 127-134, May 1976.
- 3 IRVING, T.S., SHIVA, S.G. and NAGLE, H.T.: "Flip-flops for multiple-valued logic". IEEE Trans. on Comp., v. C-25, pp. 237-247, 1976.
- 4 MOUFTAH, H.T. and JORDAN, I.B.: "Design of ternary COS/MOS memory and sequential circuits". IEEE Trans. on Comp., v. C-26, pp. 281-288, 1977.

- ⁵ HUERTAS, J.L., ACHA, J.I. and CARMONA, J.M.: "Design and implementation of tristables using c.m.o.s. integrated circuits". Electronic Circuits and Systems. Vol. 1, n° 3, Abril 1977, pp. 88-94.
- ⁶ DANIELSSON, P.E.: "Boolean memories". IEEE Trans. on Comp., v. EC-15, pp. 29-35, 1966.
- ⁷ MATTOX, J.: "Bigger n-flops". Electronic Design. v.22, n° 21, pp. 94-101, 1974.
- ⁸ UNGER, S.H.: "A study of asynchronous logical feedback networks". Research Lab. of Electronics, M.I.T., Tech. Dept. 320, June 1957.
- ⁹ SU, S and A.SARRIS: "The relationship between Multivalued Switching Algebra and Boolean Algebra under Different Definitions of Complement". IEEE Transactions on Computers, vol. C-21, n° 5, May 1972, pp. 479-485.

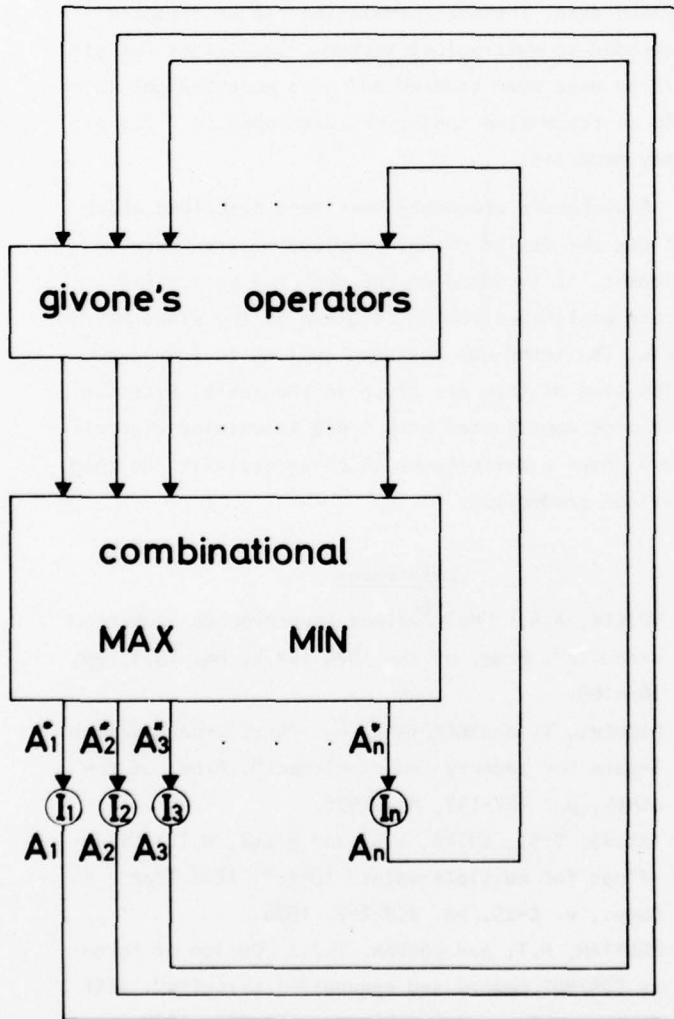


FIGURE 1: General structure for a PVME.

σ_i	I_0	I_1	$I_2 \dots$	I_m
σ_1	σ_1	σ_1	$\sigma_2 \dots$	σ_m
σ_2	σ_2	σ_1	$\sigma_2 \dots$	σ_m
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots
\vdots	\vdots	\vdots	\vdots	\vdots
σ_m	σ_m	σ_1	$\sigma_2 \dots$	σ_m

TABLE 1: Transition table for a Memory Machine.

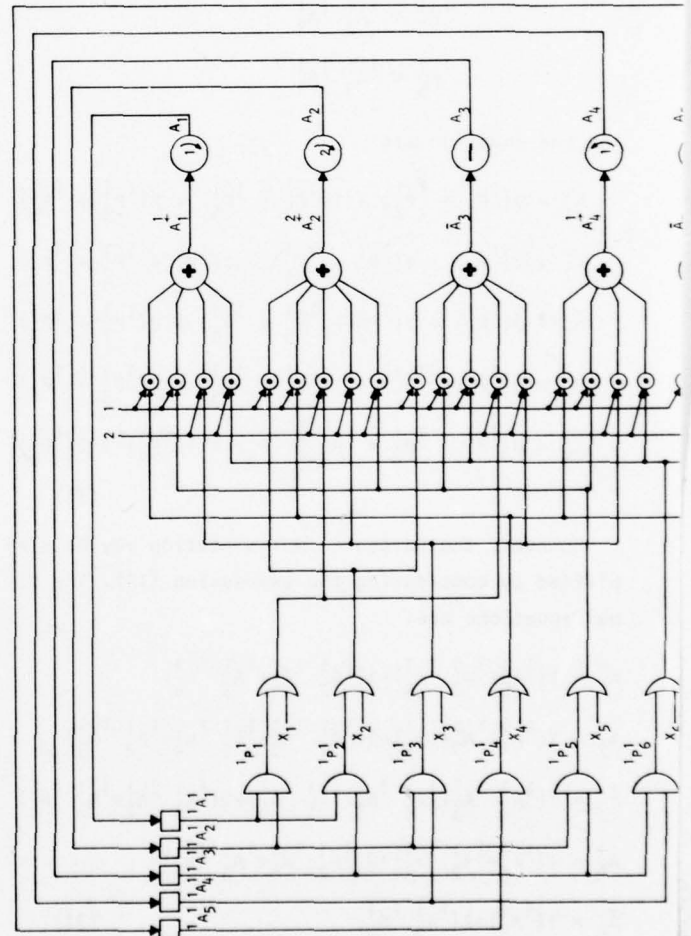


FIGURE 2: Hardware implementation of the example cited in section VII.

A TERNARY J - K MEMORY *

M A H ABDUL - KARIM

* UNIVERSITY OF BAGHDAD , IRAQ .

Indexing terms : Flip Flops , Integrated memory circuits , Integrated logic circuits , Many valued logic .

Abstract : A description of a ternary J - K type flip - flop is presented . The output is in the form of three valued level 2, 1, 0 . The circuit may be set to any logic level as well as retain the present , past and former past information in accordance to a defined J - K input logic level . This type of triflops have been selected in order to have flexible to ternary memory with simple excitation equations .

1. Introduction

Recent articles has shown much interest in the theory and design of multivalued (m.v.) memory circuits¹⁻⁶ . Many approaches and different realisation schemes have been suggested . The excitation equations of binary D, T, R - S and J - K flip - flops are used in the design of the multi - stable types . The description of a 3 valued memory elements is found in the literature 3,4,5 .

Ternary means an element of a switching which will perform 3 - valued transmission and 3 valued switching . A 3 valued variable 'Q' will assume three values Q, \bar{Q} , and $\bar{\bar{Q}}$, where the bar sign signifies cycling . Some authors such as Yofli and Rosenfeld and Porat⁷ have suggested (2,1,0) as the logic states for a ternary variable . This paper describes a J - K tristable with analogy to the binary system memory J - K flip - flop .

Integrated circuits , a few external resistors and transistors are used throughout .

2 Basic Circuit Design

The basic circuit design relies on the implementation of translating circuits between the ternary and binary set . Ternary output is obtained with the aid of a combining circuit . The logic states are represented by voltage levels as given below :

$$2 \equiv \geq 3 \text{ volt}$$

$$1 \equiv 3 \text{ volt} \geq v \geq 1.5 \text{ volt}$$

$$0 \equiv \leq 1.5 \text{ volt}$$

It was required that the J - K tristable should have the following :

- (a) J - K data input .
- (b) Clocked input .
- (c) Preset inputs , SET 0 , SET 1 , SET 2 ,
- (d) Retain present , past or former past information in accordance to a defined J - K input data .

Recently some of the authors such as Acha and Heurtas¹ have defined the concept of JK m.v . memory element and have discussed the advantages of this flip - flop . For ternary systems the JK flip - flop is defined as follows : a JK 3 valued flip - flop is one which has at least one 3 valued output and three valued inputs (J0 , J1 , J2) with the excitation table which is shown in Table 1 .

As can be seen from Table 1 , the input Jm has to be n if a transition from state m to state n must occur . It is not necessary to specify more than one input at a time .

Table 1 JK triflop excitation table

y	y ⁺	J0	J1	J2
0	0	0	x	x
0	1	1	x	x
0	2	2	x	x
1	0	x	0	x
1	1	x	1	x
1	2	x	2	x
2	0	x	x	0
2	1	x	x	1
2	2	x	x	2

This paper describes a J - K triflop whose excitation table is shown in table 2 which gives the logical operation behind this type of ternary memory .

Table 2 : Ternary J - K memory truth table

Input data		Output
J	K	Q_{n+1}
0	0	Q_n
1	0	1
2	0	0
0	1	1
1	1	\bar{Q}_n
2	1	2
0	2	0
1	2	2
2	2	$\bar{\bar{Q}}_n$

where Q_n and Q_{n+1} are the present and next states respectively . Table 2 has been arranged in analogical manner to the J - K binary flip - flop truth table . For example , when JK = 00 , the next state will be as that of the present one . Instead , for JK = 11 , it will be the complement or the cyclic shift of the present state . Similarly , for JK = 22 , a double cycling is achieved for the next transition . The combination of JK = 02 or 20 will give 0 as the next state , while for JK = 01 or 10 it will be 1; and for JK = 12 or 21 , Q_{n+1} will be at logic 2 .

3 Practical Realisation

There are three outputs represented by Q , \bar{Q} , and $\bar{\bar{Q}}$; where :

$$\bar{Q} = Q + 1$$

$$\bar{\bar{Q}} = \bar{Q} + 1$$

i .e. the three outputs are in cyclic shift mode to one another . Fig. 1 shows the basic building blocks of the J - K ternary memory . It consists of six stages : (1) Input decode (2) Input logic (3) Storage (4) Set logic (5) Output logic (6) Output encode .

3.1 Input Decode

This is achieved by means of two analog comparators (uA 710) for each input data (J or K) and in accordance to table 3 and Fig. 2 .

Table 3 Input decode table

J (or K)	a	b	J (or K)	a
0	0	0	0	0
1	1	0	1	1
2	1	1	2	x

where x denotes don't care condition .

3.2 Storage

The storage is achieved by means of two D - type flip - flops , two being the minimum number required to store the information presented to them by the input decoding circuit .

3.3 Preset Facilities

It was required to have preset facilities that will enable the Q output to take on the logical value 0 , 1 , or 2 in accordance to a single pulse application at the S_0 , S_1 or S_2 terminals respectively . The \bar{Q} and $\bar{\bar{Q}}$ outputs remaining in cyclic shift mode with the Q output . The preset inputs for the tristable are denoted as S_0 , S_1 , and S_2 while the set and reset inputs to the two D flip - flops are denoted as S_C , R_C , S_D and R_D . Following the preset truth table shown in table 4 below ,

Table 4 Preset truth table

Preset	C	D
S_0	0	1
S_1	1	0
S_2	0	0

We can write the following logical relationship :

$$S_0 = R_C \cdot S_D$$

$$S_1 = S_C \cdot R_D$$

$$S_2 = R_C \cdot R_D$$

Hence :

$$S_C = S_1$$

$$R_C = S_0 \cdot S_2$$

$$S_D = S_0$$

$$R_D = S_1 \cdot S_2$$

from which the preset logic is shown in fig. (3).

3.4 Output Encode

The D type flip - flops binary outputs are shown in the truth table (table 5) .

Table 5 Storage truth table

Logic level	C	D	E	F	G	H
0	x	1	1	0	0	0
1	1	0	0	0	x	1
2	0	0	x	1	1	0

There are at the output encoder section three circuits for the same components that translate binary data into ternary one . These circuits receive from the output logic six inputs C , D , E , F , G , and H which are fed to output encode circuitries . Fig. 4 shows the complete schematic diagram for the ternary J - K memory employing TTL gates .

4 Practical Results

The J - K triflop described above have been constructed and tested . Fig. 5 shows the output and input clock waveforms of a modulus - 3 counter for clock frequencies of 10 KHz and 1 MHz respectively .

5 Conclusions

The use of TTL and integrated circuits in the realisation of 3 - valued J - K memory elements has been discussed . The design of variety control input ternary memory is implemented .

Practical results are given that shows much interest in the versatility and speed of the implementation which have been studied . Irving et al⁴ have pointed out that m.v. logic circuits may increase their interest in the future because they allow an increase of the information content per signal line . That means a reduction of the number of interconnections . This will have the advantages of lower cost , better reliability due to lower number of interconnections and higher speed .

References

1. HEURTAS , J.L , ACFA , J.I . and CARMONA , J.M. : ' Design and implementation of tristables using c.m.o.s. integrated circuits Electronic Circuits and Systems , 1977 , 1 (3) , pp. 88 - 94
2. RATH , S.S. : ' A ternary Flip - Flop circuit , INT. J. ELECTRONICS 1975 , 38 , (1) , pp. 41 - 47 .
3. VRANESIC , Z.G. and SMITH K.G. : Engineering aspects of multi - valued logic systems ; computer , 1974 , 7 , pp. 34 - 41 .
4. IRVING , T.A., SHIVA , S.G. , and RAGLE , H.T. : ' Flip - Flops for multivalued logic ' , IEEE Trans. 1976 , 25 , pp. 237 - 26 .
5. BRUSETA , A ., VRANESIC , Z.G. and SEDRA , A.S. : ' Application of multi - threshold elements in the realisation of many valued logic networks ' , IEEE Trans. 1974 , C - 23 pp. 1194 - 1198 .
6. YOELI , M. and ROSENFELD , G. , : ' Logical design of ternary switching circuits ' , 1965 , IEEE Trans. on electronic computer , 14 , pp. 19 - 29 .
7. PORAT , D.I. : ' Three - valued digital systems ' , Proc. IEE , 1969 , 116 , (6) , pp. 947 - 954 .

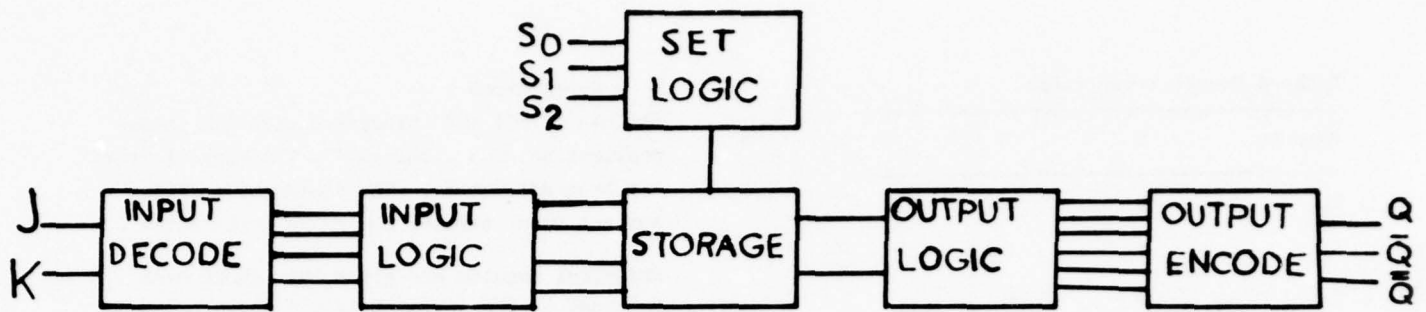


FIG1 BLOCK DIAGRAM OF THE JK TRISTABLE

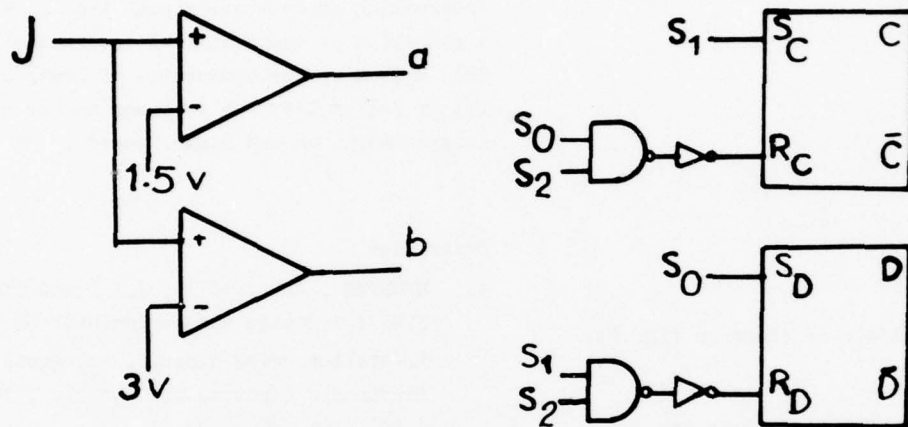


FIG 2 INPUT DECODING CIRCUIT FIG3 SET LOGIC FOR THE JK TRISTABLE

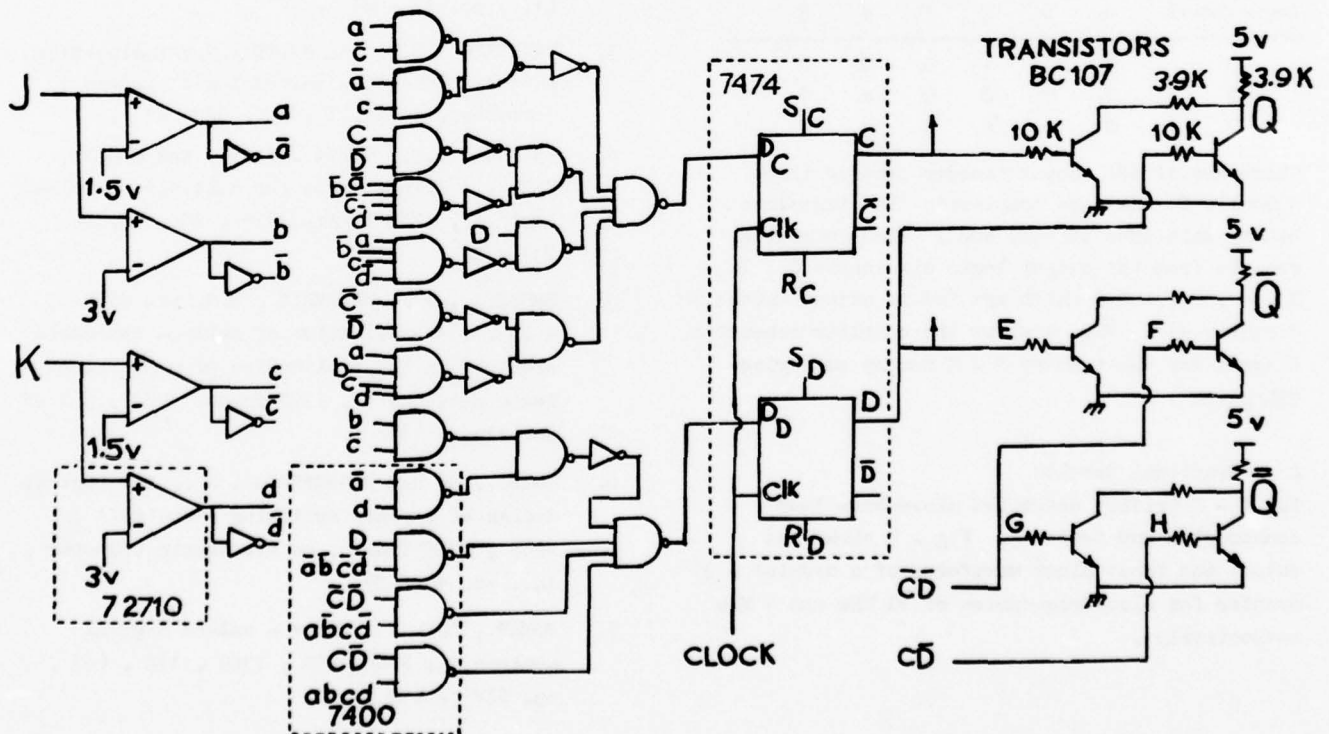
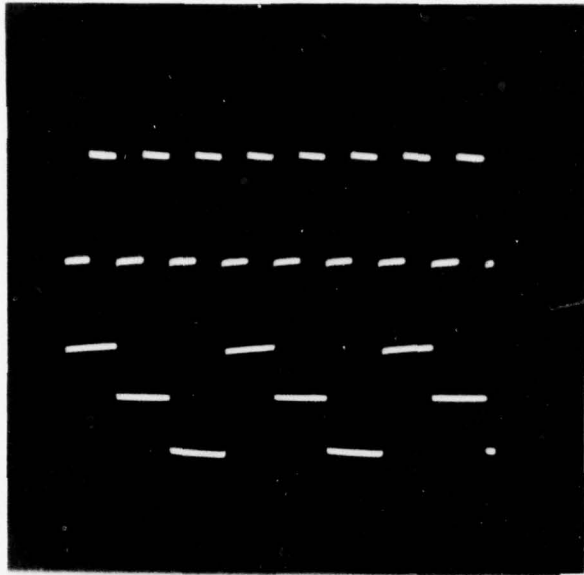
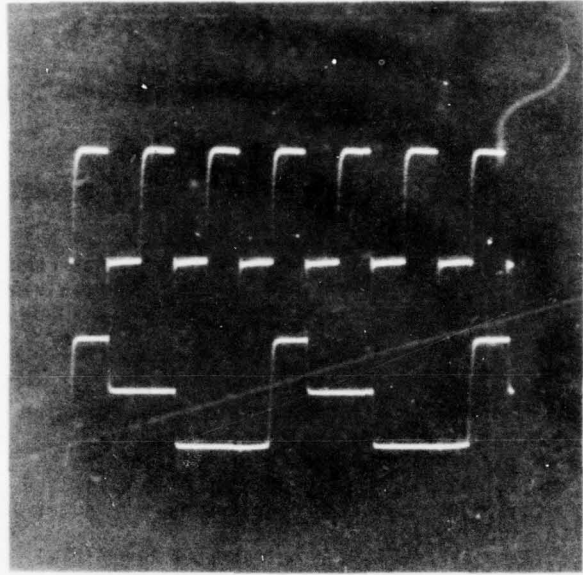


FIG 4 SCHEMATIC DIAGRAM OF A TERNARY JK MEMORY EMPLOYING TTL GATES



(a)

$2V$
 $100\mu s$



(b)

$2V$
 $1\mu s$

FIG 5 CLOCK AND OUTPUT WAVEFORMS OF THE JK TRISTABLE
 WITH A CLOCK FREQUENCY OF (a) 10KHZ (b) 1MHZ

A BEHAVIORAL MODEL AND TRIGGERING MODES FOR MVL R-FLOPS

Michael S. Wills

U. S. Air Force Systems Command
Sunnyvale AFS, California

A variety of multi-valued logic R-flop designs have been published to date, most of which were relatively ad hoc in nature, and were not based on any underlying theory of multistable behavior. Several of these designs also exhibited incorrect triggering, if proper response to intermediate signal values was required.

This paper develops a multistable behavioral model and triggering mode definitions based on observations of the binary flip-flop. It uses this model, along with other observations on R-flop behavior, to design several R-flops, and to use these R-flops in counter and shift register designs.

The Binary Flip-Flop as a Starting Point

Since the design criteria, hazard analysis, and implementation of binary flip-flops has been well researched and developed, it seems reasonable to start with these devices to consider the design of non-binary R-flops. A study of the binary flip-flops in use today will reveal three basic conclusions: that the basis for virtually all flip-flops is the cross-coupled Set-Reset type, which exhibits an unspecified state transition problem; that all of these devices can be characterized as either leading edge, level, trailing edge, or master-slave triggered devices; and that a "nested design" approach has been taken in designing other binary flip-flops, not only to eliminate the unspecified state problem but also to provide the desired selection of steering and triggering functions.

Basic Multi-Valued Logic Operators and Functions

The majority of R-flop designs have been based on the Post chain of the desired length. Many of these designs have used the familiar MIN (greatest lower bound), MAX (least upper bound), and cyclic negation ($x = r - x - 1$ modulo r) operators, along with one of a variety of other functions to make this set of operators functionally complete. In this paper, we will use the projection or coordinate functions, described by Wojcik (1,2), and defined thus²:

For an r -valued lattice, denoted $P(r)$, with elements $\{0, 1, \dots, r-1\}$ such that $0 \leq x \leq r-1$, and for $x \in P(r)$, define

$$x = (a_{r-1}a_{r-2}\dots a_1a_0) = \begin{cases} a_0 & \text{if } x=0 \\ a_1 & \text{if } x=1 \\ \vdots & \vdots \\ a_{r-1} & \text{if } x=r-1 \end{cases}$$

These operators will form the principle set of operators we will see when examining the R-flop proposals, although Druzeta demonstrates a different enhancement in the form of the MT(R) gate which can, at times, be more versatile than the projection functions.

Fundamental Characteristics of Cross-Coupled R-Flop Designs

Since the majority of the work done to date on R-flop design has centered about the cross-coupled multivibrator design, this section will summarize the more salient points of this research.

Table 1 shows the generalized next-state table for a simple R-flop of RS type, as exhibited by many researchers. As in the binary RS flip-flop, there are certain input/current state combinations that will cause the device to act in what is considered as an unstable or indeterminate fashion, especially if we insist that the two outputs of the device should be relative complements of each other. Sheafor has shown that this device can be used quite well in ternary systems, and his analysis suggests that the device will also function properly in higher radix systems as well. He further observes that any series of input transitions is valid when the corresponding output transitions vary in a monotonic non-increasing or non-decreasing fashion only³. Several other authors have considered these aspects of the cross-coupled Max-inverter or Min-inverter R-flops, obtaining essentially the same results.

One other aspect of multi-valued device behavior must be considered at this point. In all conceivable physical implementations of a multi-valued gate or more complex device, the input to the device must pass through all intermediate values between an input at some time t and that at some next-event time $t+\Delta t$. Since these

Q	RS						10	11	12	...	1r-2	1r-1	20	21	...	2r-3	2r-2	2r-1
0	0	0	0	0	...	0	1	1	1	...	1	-	2	2	...	2	-	-	
1	1	1	1	1	...	0	1	1	1	...	1	-	2	2	...	2	-	-	
2	2	2	2	2	...	0	2	2	2	...	1	-	2	2	...	2	-	-	
...																		
r-2	r-2	r-2	r-3	...	0		r-2	r-2	r-3	...	1	-	r-2	r-2	...	2	-	-	
r-1	r-1	r-2	r-3	...	0		r-1	r-2	r-3	...	1	-	r-1	r-2	...	2	-	-	

	r-20	r-21	r-22	...	r-2r-2	r-2r-1	r-10	r-11	...	r-1r-2	r-1r-1
0	r-2	r-2	-	...	-	-	r-1	-	...	-	-
1	r-2	r-2	-	...	-	-	r-1	-	...	-	-
2	r-2	r-2	-	...	-	-	r-1	-	...	-	-
...											
r-2	r-2	r-2	-	...	-	-	r-1	-	...	-	-
r-1	r-1	r-2	-	...	-	-	r-1	-	...	-	-

Table 1. Generalized Next-State Table for RS Type R-Flop

intermediate values will be present, designers are faced with two alternatives: either insist that all signal rise and fall times are much faster than the fastest settling time of any device in the system, or require all devices to respond properly to such intermediate values. The first seems an unreasonably stringent requirement and seems to demand that device output stages swing over wide value ranges faster than device input stages can respond. This alternative also places heavy restrictions on device behavior in asynchronous environments. The second alternative not only is more logically consistent, and provides for easier asynchronous use, but also seems to be easier to implement. Note, however, that a judicious use of projection or coordinate functions can simulate signals without intermediate values.

Current R-Flop Design Proposals

A number of the R-flop designs published in the current literature were reviewed in (4). Two of these reviews will be discussed here, since they provide background for the rest of this paper. Irving, Shiva, and Nagle (5-7) published several master-slave designs based on the cross-coupled RS R-flop. Druzeta (9, 10) described several devices, both single stage and master-slave, based on single loop, single active element, multi-threshold gate designs. Both Druzeta's master-slave R-flops and the ones described by Irving et. al. failed to trigger properly if clock rise and fall times were long enough to allow all gates in the devices to respond properly to all intermediate clock and data values. It was observed that the cause of this false triggering, which resulted in

level rather than master-slave triggering, was due to the slave being clocked with the complement of the master section's clock. Both authors, however, cited experiments in which devices were constructed that behaved properly even at clocking speeds exceeding three megahertz. It is unclear whether these results were merely successful attempts at tuning the circuit's response so as to ignore intermediate values, or whether these results indicate that such a rigorous treatment of intermediate values is not required for proper device behavior.

A Proposed Behavioral Model for R-Flops

The designs reviewed in (4) all seem to have one thing in common: the devices were designed based on either a new type of circuit element, or merely as attempts to extend the traditional binary flip-flop concept, without first extending the behavioral models that underly the binary flip-flop itself. Note that we distinguish the behavioral model from the next-state or excitation equations in the same fashion that the JK and JK master-slave binary flip-flops can be differentiated: both have the same next-state and excitation equations, but the triggering behavior of each is fundamentally unique. Further, there seems to be a need for a model of R-flop behavior that takes into account the "proper" handling of intermediate input, output, and clock values, and that provides for "proper" gating of new outputs to subsequent stages of logic in a total systems design. The following is proposed as such a model for R-flop behavior.

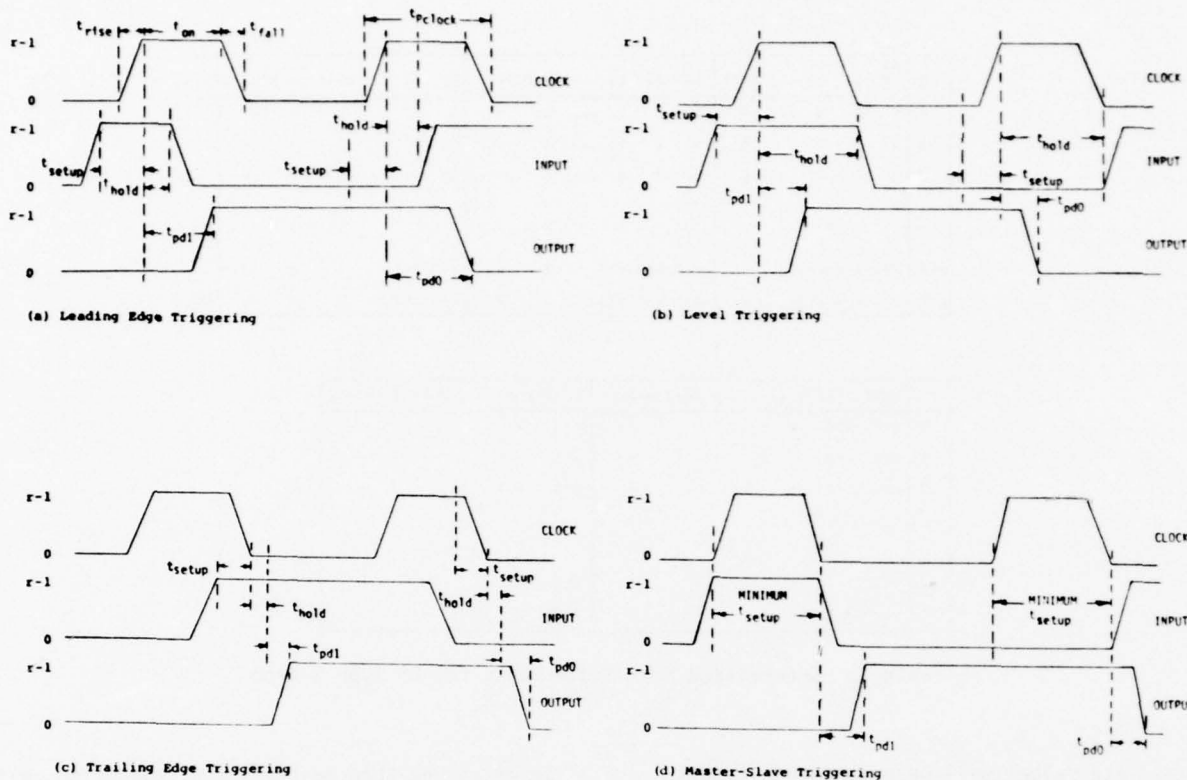


Figure 1. Multi-valued (Post Lattice) Triggering Waveforms

R-Flop Behavioral Model

An R-flop (synonyms: R-stable, multistable) is a device which:

1. is defined for any $R \geq 2$;
2. is capable of remaining indefinitely in any one of R discrete stable states when no external stimuli are applied;
3. can be described by some deterministic next-state and/or next-output function or table;
4. changes to its proper next-state value in a manner that can be described as either leading edge, level, trailing edge, or master-slave triggered;
5. presents one or more outputs that change from their present state to their next-state values in a monotonic, non-increasing (or non-decreasing), non-return-to-zero fashion;
6. can be driven from any one of its stable states to any other stable state by the application of exactly one input change (and clocking cycle, for synchronous designs);
7. responds to intermediate values of input and/or clock signals in such a fashion that is signal rise or fall time independent (within practical limits).

This model is based on a similar description by Irving and Nagle (5), but has been significantly enhanced by the inclusion of the triggering

constraints and the constraints on the manner in which output values may change, aspects not explicitly covered by a next-state equation. Other aspects of R-flop design that are desirable, but do not directly enter into the behavioral model, would include ease of nestability of design and a next-state function and total behavior that would allow R-flops to be directly substituted for binary flip-flops in register, counter, or sequential circuit designs. This would allow the wealth of binary sequential systems design techniques and experience to be used with appropriate alterations in a multi-valued environment.

Triggering Modes

The familiar binary triggering modes provide an excellent starting point for the development of R-flop triggering models, since any clocked device can be described as a leading edge, level, trailing edge, or master-slave triggered type device. Figure 1 shows these four triggering modes expanded to their R-valued forms, and detailed descriptions of each mode are as follows:

Leading Edge. The outputs of the device change to their next-state values immediately after the clock has passed the logic ($r-1$) threshold. Inputs to the device can change after

t_{hold} with no effect on the outputs of the device (until the next 0 \rightarrow (r-1) transition of the clock).

Level: The outputs change to their next-state values as soon as the clock reaches the logic (r-1) threshold. Any changes in input while the clock is high (r-1), rising, or falling will cause the appropriate change of outputs to occur.

Trailing Edge: Inputs can affect the device while the clock is rising or at the logic (r-1) state. The next-state outputs appear only after the clock has returned to a logic-0 value and not during the clock fall time.

Master-Slave: Inputs must be stable throughout the entire $t_{p(clock)}$ interval. The new output values appear only after the clock has returned to the logic-0 level. For a two-stage device (the typical form), this means that the slave section is disabled immediately after the clock leaves the logic-0 level. The master is enabled during t_{rise} and t_{on} , with the master section's outputs going to their appropriate next-state values immediately after the clock reaches the r-1 threshold. The master section is disabled and the slave begins to copy the master's outputs only after the clock has fallen to the 0 level.

Note that all input and output transitions are made in a monotonic fashion, as required by the model. Valid device outputs are guaranteed to be present only during t_{on} or t_{off} , relative to the type of triggering involved, although outputs may be valid during t_{rise} and t_{fall} for some subset of the set of all possible state transitions. Note also that these triggering definitions do not make any explicit statements regarding clock rise and fall times relative to gate or total R-flop propagation delays or settling times.

Unacceptable Input/Current-State Conditions and R-Flop Designs

As with the binary flip-flop, a "nestable" or easily-expandable design concept would be desirable. One would also like to avoid the undefined transition or unstable behavior problems inherent in RS-type binary devices and their simple multi-valued extensions.

We will now exhibit level, leading edge, master-slave, and trailing edge triggered designs, all based on the cross-coupled MIN gate Set-Reset R-flop design, that behave according to the rules presented in the previous discussion. Many authors have pointed out that the cross-coupled design has many problems with the unspecified or unstable input/current-state conditions. These transitions should not be called unstable, but rather unacceptable or unpreferred, since in fact they do hold the device in a stable, non-oscillating state wherein the Q and \bar{Q} outputs are not complements of each other. The analysis in (4) for various input/present-state values shows that a change from one acceptable input vector $RS(t)$ to another acceptable vector $RS(t+\Delta t)$ will generate the correct next-state values independent of the path taken between the two vectors. The intermediate inputs and outputs can be regarded

as the values generated before the circuit has settled to the proper next-state values. This conclusion, supported by Sheafor's results (3), clears the way to use this type of device, and its dual, the cross-coupled MAX gate device, in many applications. It further indicates that these devices will support the requirement that any valid state can be reached from any other valid state by the application of one change of input vector from one acceptable value to another. Figure 2 shows this RS R-flop and its next-state table in a generalized form, with unacceptable or unpreferred transitions indicated with a dash and stable states indicated with an asterisk under the next-state entry. This table was derived from simulation results for radices of size two through six, described in (4), and shows that so long as inputs that would result in an unacceptable final result are avoided, the device will function in a stable, glitch-free fashion.

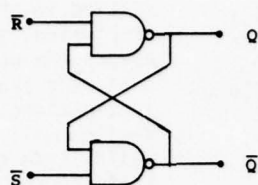
However, if one attempts to drive the device to an unpreferred state (one where \bar{R} and \bar{S} are not complements of each other, or where the resulting outputs are not complementary), then nondeterministic behavior results. Figure 3 shows a transition diagram for a four-valued example where it was attempted to drive the device with $RS=11$. Thus, owing to this device's peak-detecting nature (described in the next section), one must insure that the \bar{RS} inputs never go below the actual values desired to set the device into one of its r proper states. (Note that this also agrees with the behavioral model's constraint that the R-flop should have exactly r stable states.) The weight of the available evidence, both theoretical and practical, seems to suggest that this device can be used safely in R-flop designs that avoid these unspecified or nondeterministic final states.

Level Triggered R-Flops

A level triggered RS R-flop is shown in Figure 4. This device is not truly level triggered, however, in that while the clock is on, changes in the RS inputs to values greater than those values initially present will cause the device to change state, but changes to values less than the initial values will not. This indicates that this R-flop design has a previously unreported "peak detecting" property as well as its properties as a storage device. Most of the devices proposed to date, as discussed earlier in this paper, turn out to be level triggered devices. Most foreseeable applications, however, will require more sophisticated triggering as well as more versatile next-state functions than the RS model provides.

JK Trailing, Master-Slave, and Leading Edge Triggered R-Flops

Figure 5(a) shows the next-state table for a ternary JK type R-flop as described by several authors. Consider this device operating as a level or trailing edge triggered device, which allows its outputs to change state during the clocking interval. This can allow the device to operate improperly, as the transition graph of Figure 5(b) shows. From this, we can see that the



RS																																																																	
Q \overline{Q}	00	01	02	...	0Y	0Z	10	11	...	1X	1Y	1Z	20	21	...	2W	2X	2Y	2Z	30	31	...	3V	3W	3X	3Y	3Z																																					
0	ZZ	ZY	ZX	...	Z1	Z0	YZ	YY	...	Y2	Y1	Y1	XZ	XY	...	X3	X2	X2	X2	WZ	WY	...	W4	W3	W3	W3	W3																																						
	-	-	-		-		-	-		-			-	-		-				-	-		-																																										
1	ZZ	ZY	ZX	...	Z1	Z0	YZ	YY	...	Y2	Y1	Y1	XZ	XY	...	X3	X2	X2	X2	WZ	WY	...	W4	W3	W3	W3	W3																																						
	-	-	-		-		-	-		-			-	-		-				-	-		-																																										
2	ZZ	ZY	ZX	...	Z1	Z0	YZ	YY	...	Y2	Y1	Y1	XZ	XY	...	X3	X2	X2	X2	WZ	WY	...	W4	W3	W3	W3	W3																																						
	-	-	-		-		-	-		-			-	-		-				-	-		-																																										
.																																																																	
.																																																																	
.																																																																	
X	ZZ	ZY	ZX	...	Z1	Z0	YZ	YY	...	Y2	Y1	Y1	XZ	XY	...	X3	X2	X2	X2	WZ	WY	...	W4	W3	X2	X2	X2																																						
	-	-	-		-		-	-		-			-	-		-		*	*	*	-	-		-		*	*	*																																					
Y	ZZ	ZY	ZX	...	Z1	Z0	YZ	YY	...	Y2	Y1	Y1	XZ	XY	...	X3	X2	Y1	Y1	WZ	WY	...	W4	W3	X2	Y1	Y1																																						
	-	-	-		-						*	*	-	-				*	*	-	-					*	*	*																																					
Z	ZZ	ZY	ZX	...	Z1	Z0	YZ	YY	...	Y2	Y1	Z0	XZ	XY	...	X3	X2	Y1	Z0	WZ	WY	...	W4	W3	X2	Y1	Z0																																						
	-	-	-		-	*				-		*	-	-		-			*	-	-		-				*																																						

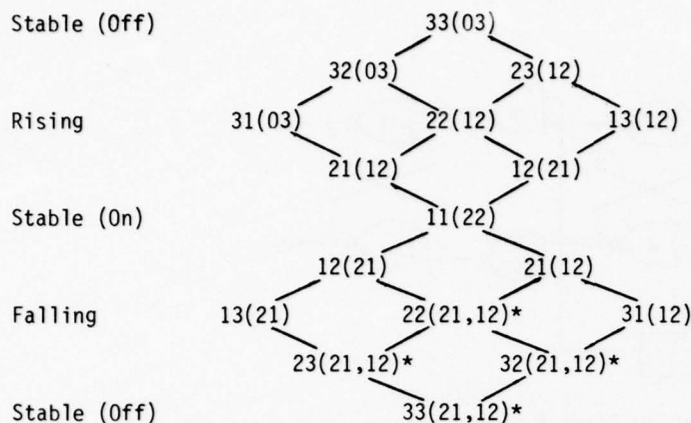
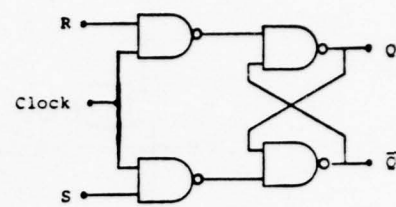
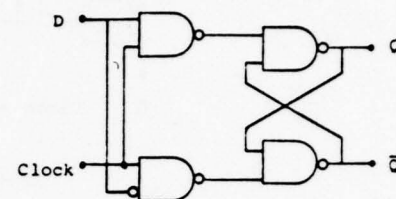


Figure 3. Transition Graph for $\overline{RS}=11, \overline{Q_0}=03$.
(Note: each node of form $\overline{RS}(\overline{QQ})$.)



(a) RS type

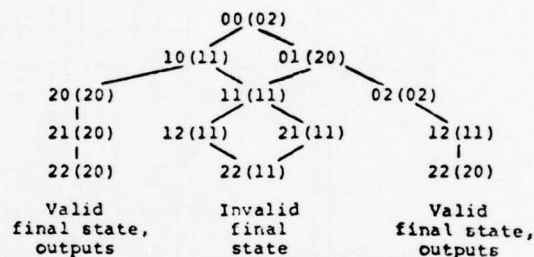


(b) D type

Figure 4. Simple Level Triggered Clocked R-Flops

Q	JK			10 11 12			20 21 22		
	00	01	02	10	11	12	20	21	22
0	0	0	0	1	1	1	2	2	2
1	1	1	0	1	1	1	2	1	1
2	2	1	0	2	1	0	2	1	0

(a) Next-state table



(b) Transition graph, $Q_0 = 0, JK = 22$ (toggle)

Figure 5. Hazard-Prone JK Trailing Edge Triggered R-Flop

JK R-flop must be a master-slave triggered device, a conclusion supported by other authors as well. Figure 6 shows a JK master-slave R-flop, which is similar to the device proposed by Irving, et. al., except for the way in which the slave section is clocked. A projection function is used to clamp the slave's clock to either r-1 or 0, isolating the slave from the master immediately after the external (master) clock goes from 0 to r-1. This device will also behave as a trailing edge triggered peak-detecting device, if the setup time constraints are relaxed somewhat.

From the detailed analysis in (4) it is concluded that this proposed JK master-slave design will function properly without regard to clock rise and fall times, while the clock on time need be only so long as the master section's input gating delay and settling time. The same master-slave design can also be used without the steering network to produce an RS master-slave design, or with an inverter to produce a D master-slave, illustrating the nestability of this R-flop design technique. Note also that a properly functioning D-type R-flop such as Sintonen's (8) with its possible race conditions removed can also be used as the master and slave stages with equal results

when clocked as described above.

In the same fashion that a binary master-slave can be used as a leading edge device by externally inverting the clock, the R-flop of Figure 6 can be transformed into a leading edge device. This design will allow multiple input changes while the external clock is off, unlike some earlier binary master-slave devices, since the master section will follow each change as it occurs, while the slave is locked out at this time. This behavior further supports the argument given above that this design can function as a trailing edge triggered device, although its peak detecting nature must be reckoned with for designs that call for multiple input changes during a particular clocking cycle.

Application Designs Using the Proposed R-Flop

To test the ease with which binary flip-flop applications may be generalized to produce multi-valued versions, the JK master-slave R-flop exhibited in the previous discussion will be used in two typical binary applications expanded to multi-valued designs.

The first application example is that of a

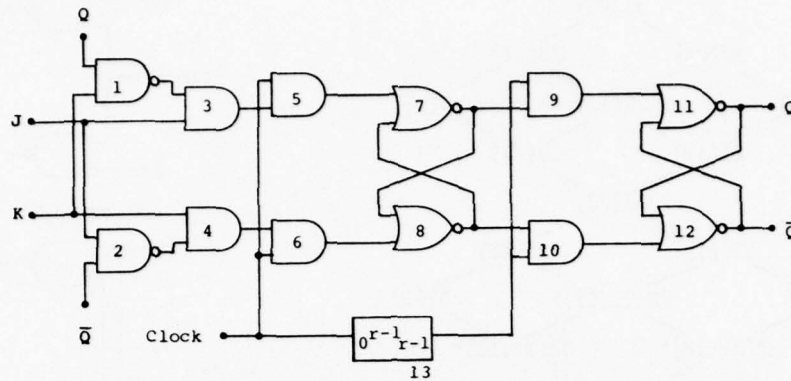
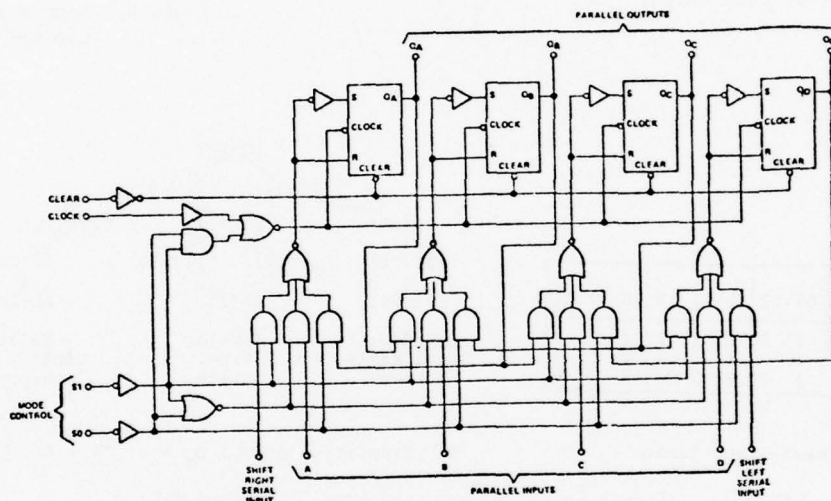
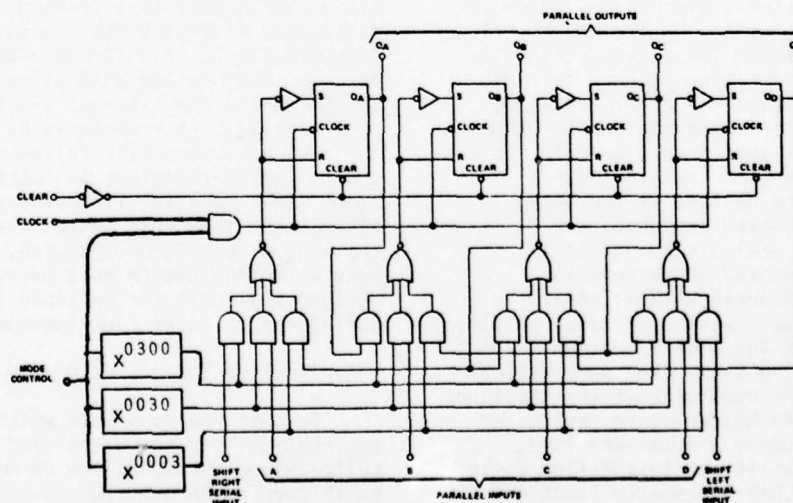


Figure 6. JK Master-Slave R-Flop



(a) Type 74194 Binary Shift Register



(b) Four-Valued MVL Shift Register

Figure 7. Bidirectional Shift Register Designs

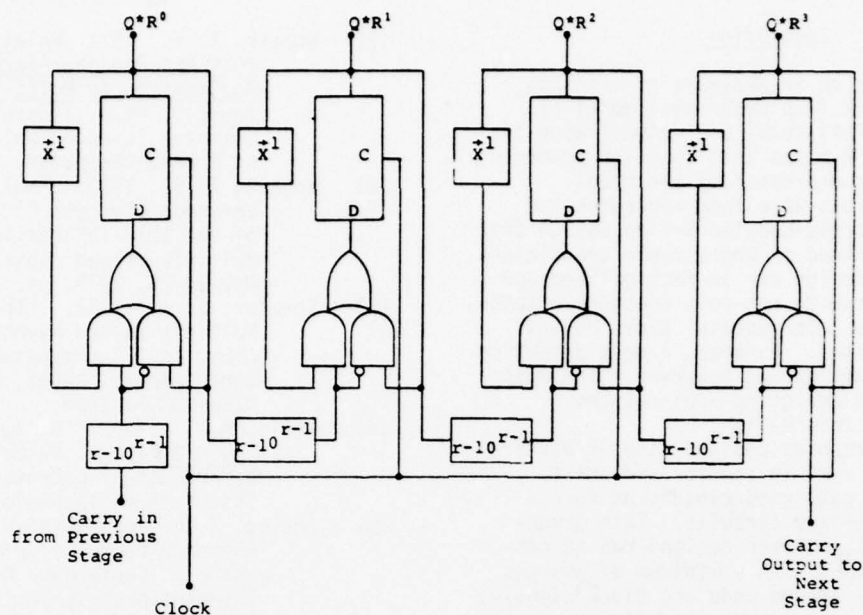


Figure 8. MVL 4 Stage Up Counter Design

four-stage bidirectional parallel-in, parallel-out shift register. This design will be carried out for a four-valued system, and will use the logic design of the type 74194 TTL integrated circuit shift register, shown in Figure 7(a). The four-valued functionally equivalent design is shown in Figure 7(b). The changes that were needed centered on the clocking and mode control circuitry; the actual flip-flop interconnections and input gating needed no redesign at all. A single four-valued input signal is used for mode control, with values of 0, 1, 2, and 3 corresponding to shift left, parallel load, shift right, and "do nothing" functions respectively. Ramp (projection) functions are used to decode the mode signal into the appropriate binary enable/disable function controls. The details of operation can be found in (4).

Shift or storage register designs are probably the easiest R-flop applications one can consider. The design of a synchronous multi-values counter, however, is not quite as straightforward, for instead of merely toggling each successive stage in the counter, a multi-valued counter needs to add 1 to the contents of each successive stage at the moment its predecessor goes from the $r-1$ state to 0. Most authors who have proposed MVL counter designs have used a cycle function to provide this "add 1" action. A further observation that should be made at this point is that MVL counter designs, as well as shift registers, should only use master-slave triggered R-flops internally. Figure 8 shows a design that directly clocks all R-flops but uses ramp functions to detect the $r-1$

to 0 transition of the n_{th} stage's output to select either the cycle gate's output or the Q_{n+1} output as the D_{n+1} input. This design will function synchronously, since there is no relative delay between the clock signal for all of the R-flops, and since the value of Q_n and Q_{n+1} that are present while the clock is off (and that remain stable during the R-flop setup time) are the ones that are used to trigger the $n+1$ st stage. Changes to the values of Q_n or Q_{n+1} while the clock is high or falling will have no effect on the state of the $n+1$ st stage during the clocking cycle.

One of the undesirable aspects of this kind of counter design is its heavy dependance upon the cycle and ramp function gates. This would be undesirable if these functions turn out to be substantially more costly to implement reliably than simple MAX, MIN, and inverter gates. But since MAX, MIN, and complement functions do not form a functionally complete set by themselves, either one or both of the ramp or cycle operators will need to be developed in a cost-effective fashion. The generalized form of the ramp function can be used to implement the cycle operator (eg. $x(r-1+a, r-2+a, \dots, 1+a, a)$ for fixed values of a), and indeed looks much like the threshold functions of Druzeta's MT(R) gates, so it does not seem too unreasonable to expect fast, cost-effective electronic realizations of either the cycle function, specific ramp or coordinate functions, or the fully generalized projection

functions.

Conclusion

Based in part on an analysis of existing R-flop designs an R-flop behavioral model has been developed. This model was demonstrated by designing several R-flops that function according to the constraints expressed by the model. Detailed observations were made regarding the operation of the cross-coupled R-flop design that indicate the undefined or undesirable transitions inherent in this design are in fact defined and stable but give rise to non-complementary outputs, and can be used as intermediate values during device settling times. Further, a peak detection property of this design was observed which must be considered when designing applications circuits that use this R-flop.

One of the proposed designs, the JK master-slave R-flop, was used in counter and shift register circuits patterned closely on functionally similar binary circuits. This demonstrated that shift register designs can be converted to MVL designs with a minimum of changes centered primarily around mode and clock signals, which still tend to be binary (on/off) in nature, even in MVL systems. Counter designs, on the other hand, are shown to require the use of the cycle operator for each stage of the counter, and fully synchronous designs seem to require the use of projection functions to properly detect the $(r-1)$ 0 transitions of each stage in order to trigger its successor stage. One projection

function in particular, $x^{((r-1)0^{r-1})}$ (and its dual $x^{(0^{r-1}(r-1))}$) is shown to be of great utility in eliminating clocking difficulties in R-flop designs, and is especially useful in applications where undisciplined clock signals may be present and could cause spurious triggering to occur.

While this paper has taken a strong position on the subject of proper response to intermediate clock and data signals, it has made no real attempt to resolve the issue. However, there does seem to be some experimental evidence supporting a more relaxed view of the intermediate value response issue. Theoretical results such as those presented here and in (4) still show that a circuit or system must respond to each value a signal takes on at its inputs regardless of the duration of that signal. Further research both on the theoretical and engineering aspects of MVL gate and device behavior would be needed to resolve this issue.

Bibliography

- (1) Wojcik, A. S. 1971 Relationships Between Post and Boolean Algebras with Application to Multi-Valued Switching Theory. Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.
- (2) Wojcik, A. S. 1973. "Multi-Values Asynchronous Circuits." Conference Record of the 1973 International Symposium on Multiple-Valued Logic. Toronto, Canada, May 24-25, 1973, pp. 217-227.
- (3) Sheafor, S. J. 1974. "The Design of Multiple-Valued Asynchronous Sequential Circuits." Coordinated Science Laboratory Report, R-663, September, 1974. UILU-ENG 74-2229.
- (4) Wills, M. S. 1977. A Study of Multi-Valued Logic R-Flops. Master's thesis, Department of Computer Science, Illinois Institute of Technology.
- (5) Irving, T. A. and Nagle, H. T. 1973. "An Approach to Multi-Valued Sequential Logic." Conference Record of the 1973 International Symposium on Multiple-Valued Logic. Toronto, Canada, May 24-25, 1973, pp. 89-105.
- (6) Shiva, S. G., and Nagle, H. T. Jr. 1974. "On Multiple-Valued Memory Elements." Proceedings of the 1974 International Symposium on Multiple-Valued Logic. Morgantown, West Virginia, May 29-30, 1974, pp. 209-224.
- (7) Irving, T. A. Jr., Shiva, S. G., and Nagle, H. T. Jr. 1976. "Flip-Flops for Multiple-Valued Logic." IEEE Transactions on Computers. Vol. C-25, No. 3, March 1976. Long Beach, Ca. pp. 237-246.
- (8) Sintonen, L. 1976. "A Clocked Multi-Valued Flip-Flop." IEEE Computer Society Repository. IEEE Computer Society, Long Beach, California. Report 76-107.
- (9) Druzeta, A. and Sedra, A. S. 1973. "Multi-Threshold Circuits in the Design of Multi-State Storage." Conference Record of the 1973 International Symposium on Multiple-Valued Logic. Toronto, Canada, May 24-25, 1973, pp. 49-58.
- (10) Druzeta, A. 1974. Many-Valued Multi-Threshold Logic. Ph.D. thesis, Department of Electrical Engineering, University of Toronto.

GENERALIZED FINITE POST ALGEBRAS

J. C. Muzio*
Dept. of Computer Science
University of Manitoba

and
T. C. Wesselkamper
Dept. of Computer Science
Virginia Polytechnic Institute
and State University

Abstract

A generalized Post algebra is defined and discussed. It is proved that generalized Post algebras are complete whereas Post algebras, other than those based on a single totally ordered chain, are not. Techniques for the expression of arbitrary many-valued switching functions in the generalized Post algebra are described.

1. Introduction

In this paper we are interested in many-valued functions $f x_1 \dots x_n$ defined on $E(k) = \{0, 1, \dots, k-1\}$.¹ Parenthesis free notation is used for the functions. A set A of functions over $E(k)$ is complete if it is possible to define any function over $E(k)$ as a composition of functions from A . If the union of the set A and the constant functions is complete then A is called complete with constants.

While Boolean algebras have played a huge role in two-valued switching theory, they are not generally useful in multiple-valued switching theory, since they exist only over spaces of 2^m elements, and even for these spaces they are neither complete nor complete with constants except for $m = 1$.

A functionally richer system, complete with constants, was developed by Dussault, Metze and Krieger¹. We shall refer to their structure as a generalized Boolean algebra. Briefly, they add to the usual Boolean structure a set $\{x^a\}$ of one-place characteristic functions one for each point of the space. These generalized Boolean algebras still have the drawback that they exist only for spaces of 2^m points.

* The work reported herein was performed while the first author was a visiting Associate Professor in the Department of Computer Science, Virginia Polytechnic Institute and State University.

It is well known that Post algebras as formalized by Epstein^{2,3} are only complete when based on a single totally ordered chain. In this paper we present a new, more general description of a type of Post algebra. It uses a different, rather larger, fixed set of points than Epstein, together with the characteristic functions defined at these points. We prove that this generalized Post algebra is complete and briefly discuss the expression of arbitrary functions on $E(k)$ in the generalized Post algebra. Initially we define some terminology and results, which are used in the following sections.

The context of the whole of this paper is the distributive lattice with greatest and least element. Throughout we let $\underline{1}$ denote the greatest and $\underline{0}$ denote the least element of the lattice. We call a function a decisive function if it assumes only the values $\underline{0}$ and $\underline{1}$. A two place function \cdot is called a product type function if and only if, for all x in $E(k)$, $\underline{0} \cdot x = x \cdot \underline{0} = \underline{0}$ and $\underline{1} \cdot x = x \cdot \underline{1} = x$. A two place function $+$ is called a sum-type function if and only if, for all x in $E(k)$ $\underline{0} + x = x + \underline{0} = x$. Neither a sum-type function nor a product-type function need be either commutative or associative.

Finally we state two Representation Theorems which are well known.

Representation Theorems: Any function $f x_1 \dots x_n$ of n variables over $E(k)$ can be written in the form:

$$f x_1 \dots x_n = \sum_{i=0}^{k-1} J_i x_i \cdot f i x_2 \dots x_n$$

where $+$ and \cdot are respectively sum-type and product-type functions and the J_i are characteristic functions.

Any function $f x_1 \dots x_n$ of n variables over $E(k)$ can be written in the form:

$$fx_1 \dots x_n = \sum_{i=1}^{k-1} i \cdot g_i x_1 \dots x_n$$

where $+$ and \cdot are respectively sum-type and product-type functions and the g_i are decisive functions of at most n variables.

2. The Chain-based Post Algebras

The generalized Boolean algebras of the previous section only exist for k some power of 2. In this paper we develop other generalizations of Boolean algebra. Initially there is a clear distinction between two possible generalizations, the first taking the Boolean algebra based on $E(2)$ and using that, while the second expands a Boolean algebra based on $E(2^m)$. In both cases the structures which are constructed contain an embedded Boolean algebra. It will be recalled that a Boolean algebra is a complete complemented distributive lattice. Clearly to generalize the structure some of these lattice properties must be dropped; in particular the complemented property is allowed to lapse and we only insist on a complete distributive lattice.

Specifically, we may define a chain $0 < 1 < \dots < k-1$, and upon this chain define the operation \cdot to be the min function and the operation $+$ to be the max function. Under these operations the chain forms a distributive lattice with greatest element $k-1$ and least element 0 .

Definition 1: A chain based Post algebra $\langle E(k), \mathbb{J}_1, +, \cdot, 0, 1 \rangle$ is such that

- (i) The elements of $E(k)$ form a totally ordered chain, $0 < 1 < \dots < k-1$.
- (ii) $+$ and \cdot are defined to be the max and min operations respectively.
- (iii) There is an embedded Boolean algebra of size 2 based on $\{0, 1\}$.
- (iv) \mathbb{J}_1 is the set of k characteristic functions, $J_i x$, where

$$J_i x = \begin{cases} 1 & \text{if } x = i \\ 0 & \text{if } x \neq i \end{cases}$$

As a notational convenience we write $\{0 < \dots < k-1\}$ to denote the set of elements in the totally ordered chain $0 < 1 < \dots < k-1$ and, more generally the set of elements in the totally ordered chain $b_i < \dots < b_j$ is denoted by $\{b_i < \dots < b_j\}$.

Chain based Post algebras have been studied extensively. In particular it is well known that they are complete with constants. We list below, without proof, certain useful and well-known results.

Theorem 1: Every one-place function $f : E(k) \rightarrow E(k)$ is of the form

$$fx = \sum_{i=0}^{k-1} J_i x \cdot f_i$$

Theorem 2: Any function $fx_1 \dots x_n$ of n variables over $E(k)$ can be written in the form

$$fx_1 \dots x_n = \sum_{i=0}^{k-1} J_i x_i \cdot fix_2 \dots x_n$$

where $+$ and \cdot are respectively a sum-type and a product-type function.

Theorem 3: Any function of n variables over $E(k)$ can be expressed in the form

$$fx_1 \dots x_n = \sum_{e_1=0}^{k-1} \dots \sum_{e_n=0}^{k-1} J_{e_1} x_1 \cdot \dots \cdot J_{e_n} x_n \cdot fe_1 \dots e_n$$

where $+$ and \cdot are respectively a sum-type and a product-type function.

Theorem 4: Any function of n variables over $E(k)$ can be expressed in the form

$$fx_1 \dots x_n = \sum_{i=1}^{k-1} i \cdot f_i$$

where each f_i is a decisive function of at most n variables, and $+$, \cdot are respectively a sum-type and a product-type function.

Theorem 5: If f and g are decisive functions over $E(k)$ and i, j in $E(k)$ with $+$ a sum-type function such that $i + j = j$ then $i \cdot f + j \cdot g = i \cdot (f + g) + j \cdot g$ where \cdot is any product-type function.

Since $i + j = j$ if $i < j$ for any $i, j \in E(k)$ it is always possible to use this theorem, as illustrated in Example 1.

Example 1: A two-place function $fx y$ over $E(5)$.

$fx y$	0	1	2	3	4	y
0	0	0	1	4	2	
1	2	2	1	4	4	
$x \ 2$	4	1	0	0	0	
3	3	4	0	4	0	
4	1	3	3	4	3	

$$fx y = 1 \cdot f_1 x y + 2 \cdot f_2 x y + 3 \cdot f_3 x y + f_4 x y$$

where

$$\begin{aligned} f_{1xy} &= J_1x + J_4x + J_0x \cdot J_2y + J_2x \cdot J_1y \\ f_{2xy} &= J_1x \cdot J_2'y + J_0x \cdot J_4y \\ f_{3xy} &= J_3x \cdot J_0y + J_4x \cdot J_0'y \\ f_{4xy} &= J_2'x \cdot J_3y + J_1x \cdot J_4y + J_2x \cdot J_0'y + J_3x \cdot J_1'y \end{aligned}$$

Note that $J_i'x$ is used as an abbreviation for $J_0J_i'x$, and $J_i'x = \begin{cases} 0 & x = i \\ 1 & x \neq i \end{cases}$.

3. Distributive Lattices and Generalized Post Algebras

In the previous section we considered the chain-based Post algebra which consists of the single chain. Larger Boolean algebras follow from the two-valued by taking a direct product of these chains and our generalized Post algebra similarly consists of a direct product of chains, but the chains may contain more than two points. Indeed Section 2 went from $k = 2$ to $k = q$, $\{0 < \dots < q\}$ being the points whereas here we take a Boolean algebra based on $k = 2^m$ and expand it to a structure based on $k = q_1q_2 \dots q_m$.

We develop the lattice structure of the generalized Post algebras first. Suppose we consider a complete distributive lattice $L = \langle E(k), +, \cdot, 0, 1 \rangle$ with greatest element 1 and least element 0 . Further let B be the maximal set of complemented elements in L . Any closed subset of a distributive lattice is again a distributive lattice. It is easily proved that B is a Boolean algebra provided it is closed and contains the elements 0 and 1 .

By the Stone representation theorem there is an integer m such that $B = B(2^m)$, the Boolean algebra with 2^m elements. This integer m is the dimension of the Boolean algebra.

For sizes 2 and 3 there is only one complete distributive lattice of each order, the chain. For larger sizes there is a rich collection of complete distributive lattices. Hereafter we limit our discussion to complete distributive lattices which are the Cartesian product of chains. Although the development which follows can be carried out on any complete distributive lattice, the results are only useful in the cases of Cartesian products of chains.

In general if k has a factorization $k = q_1q_2 \dots q_m$ into natural numbers, where for each i , $q_i \neq 1$, then there is a lattice L of size k with embedded Boolean algebra $B(2^m)$. This m is termed the dimension of the distributive lattice L . In the case $k = q$, this lattice is just a chain and the embedded Boolean algebra is $B(2)$. The lattice of size k with maximal dimension is obtained by choosing all q_i to be prime.

Note that the nature of the distributive lattice under consideration is characterized by the particular factorization of k , for example $12 = 3 \cdot 2 \cdot 2$. This method is used below to describe the distributive lattice. Further we insist, for convenience, that when k is factorized the factors are written in decreasing order of size; that is if $k = q_1q_2 \dots q_m$ then $q_i \leq q_j$ if $i > j$.

For the generalized Post algebra we use sets of points rather like a coordinate base on which to construct the superstructure of the lattice. These sets of points will consist of all the nonzero points which start at 0 and end at an atom of the embedded Boolean algebra.

Definition 2: A nonzero element of a distributive lattice is an atom if and only if, for all elements b in the lattice $b \leq a$ implies $b = a$ or $b = 0$.

For convenience the set M of atoms of a distributive lattice of dimension m will always be labelled $1, 2, \dots, m$.

Let us consider the position we are in. L is a complete distributive lattice of size $k = q_1 \dots q_m$ with an embedded Boolean algebra of size 2^m . A is the set of atoms of Boolean algebra and M is the set of atoms of L . Both these sets contain m elements and are not necessarily disjoint. The set of points which we use as a basis for building the lattice is the bottom generating set, G , consisting of all the nonzero points lying on chains joining 0 to atoms in A .

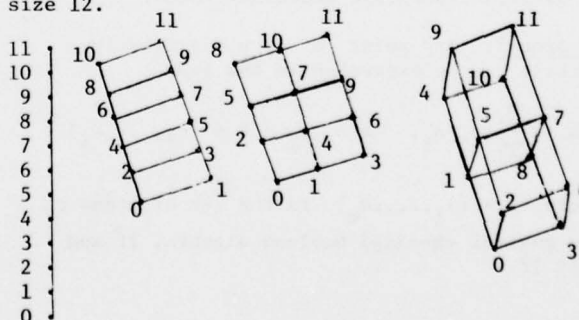
Definition 3: The bottom generating set G for a distributive lattice L of dimension m with embedded Boolean algebra B of dimension m is defined by

$$G = \{b : b \neq 0 \text{ and } b \in \{0 < \dots < a_i\} \text{ for some } a_i \in A\}$$

where $A = \{a_1, \dots, a_m\}$ is the set of atoms of B .

Note that it is also possible to define a dual top generating set. Also, for convenience, $a_i \in A$ will be the atom of B on the chain starting at 0 and including i in M .

Example 2: The four distributive lattices of size 12.



- (i) For $12 = 12$. $B = \{0, 11\}$, $A = \{11\}$, $M = \{1\}$,
 $G = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$.
- (ii) For $12 = 6 \cdot 2$. $B = \{0, 1, 10, 11\}$, $A = \{1, 10\}$,
 $M = \{1, 2\}$, $G = \{1, 2, 4, 6, 8, 10\}$.
- (iii) For $12 = 4 \cdot 3$. $B = \{0, 3, 8, 11\}$, $A = \{3, 8\}$,
 $M = \{1, 2\}$, $G = \{1, 2, 3, 5, 8\}$.
- (iv) For $12 = 3 \cdot 2 \cdot 2$. $B = \{0, 2, 3, 4, 6, 9, 10, 11\}$
 $A = \{4, 2, 3\}$, $M = \{1, 2, 3\}$, $G = \{1, 2, 3, 4\}$.

Note that if $k = q_1 \dots q_m$ the number of elements in G is given by

$$q = \sum_{i=1}^m (q_i - 1).$$

The expression of points in the lattice in terms of the elements of G is considered in the following theorems.

Theorem 6: Any point b in a distributive lattice L can be uniquely expressed in the form

$$b = \sum_{i=1}^m r_i, \text{ each } r_i \in \{0 < \dots < a_i\}$$

and $A = \{a_1, \dots, a_m\}$ is the set of atoms of the maximal embedded Boolean algebra.

Proof: Let $L = E(k)$ and $k = q_1 \dots q_m$. The number of points in $\{0 < \dots < a_i\}$ is q_i . Each expression of the type given in the theorem can be identified with an ordered m -tuple (r_1, \dots, r_m) . Each such m -tuple clearly refers to some point in L and further the number of distinct m tuples is $q_1 \dots q_m = k$. Finally it is straightforward to prove that all the k points in $E(k)$ must have a representation by showing that two distinct m -tuples must refer to distinct points.

This result gives a form of normal representation for a point, namely as the sum of elements of G , the bottom generating set. The m -tuple (r_1, \dots, r_m) are called the normal coordinates of the point. The alternative representation developed below uses the concept of different possible coordinate sets.

Theorem 7: Any point b in a distributive lattice can be expressed in the form

$$b = \sum_{i=1}^m p_i \cdot a_i, \text{ each } p_i \in P = \{e_1, \dots, e_s\},$$

where $A = \{a_1, \dots, a_m\}$ is the set of atoms of the maximal embedded Boolean algebra, if and only if

- (i) $G \subseteq \{a_i \cdot e_j : a_i \in A, e_j \in P\}$, and
(ii) To each a_i there exists some j such that $a_i \cdot e_j = 0$.

(The implication of these conditions is that there is a representation of a point using a particular coordinate set P if and only if every point of G can be expressed as a product $a_i \cdot e_j$ and further that for each atom in A there is an element in the coordinate set to map it into 0).

Proof: Let $U = \{a_i \cdot e_j : a_i \in A, e_j \in P\}$.

Initially we prove the necessity of the conditions. Suppose such a representation exists for every point b in the distributive lattice.

$$\text{Consider some } b \in G, B = \sum_{i=1}^m p_i \cdot a_i, p_i \in P.$$

Since $b \in G$, $b \in \{0 < \dots < a_j\}$ for some j .

Hence

$$b = b \cdot a_j = \sum_{i=1}^m p_i \cdot a_i \cdot a_j = p_j \cdot a_j \text{ since } a_i \cdot a_j = 0 (i \neq j),$$

and so $G \subseteq U$.

$$\text{For (ii) let } b_j = \sum_{i=1, i \neq j}^m a_i.$$

There exists some representation for b_j , say

$$b_j = \sum_{i=1}^m p_i \cdot a_i.$$

$$\text{Now } 0 = b_j \cdot a_j = \sum_{i=1}^m p_i \cdot a_i \cdot a_j = p_j \cdot a_j.$$

Hence $p_j \cdot a_j = 0$, and consequently for each a_j there exists an element p_j of P such that $p_j \cdot a_j = 0$.

This shows the necessity of the two conditions.

Conversely, suppose the two conditions hold. By Theorem 6 any point b may be expressed in the form

$$b = \sum_{i=1}^m r_i, r_i \in \{0 < \dots < a_i\}.$$

By (i) and (ii) of the theorem we can express each r_i in the form $r_i = e \cdot a_i$ for some $e \in P$. Let $p_i = e$ so that

$$b = \sum_{i=1}^m p_i \cdot a_i \text{ as required.}$$

This completes the theorem.

Corollary 8: If $k = q_1 \dots q_m$ with $q_i \geq q_j$ ($i < j$) and a representation of each point exists then P contains at least q_1 points.

Proof: Consider $b, c \in \{0 < \dots < a_i\}$ with $b \neq c$. By the theorem $b = p_1 \cdot a_i$, $c = p_2 \cdot a_i$ for some $p_1, p_2 \in P$. Clearly $p_1 \neq p_2$. Since there are exactly q_i distinct points in $\{0 < \dots < a_i\}$ there must be at least q_i distinct points in P . Since $q_1 \geq q_i$ the result follows.

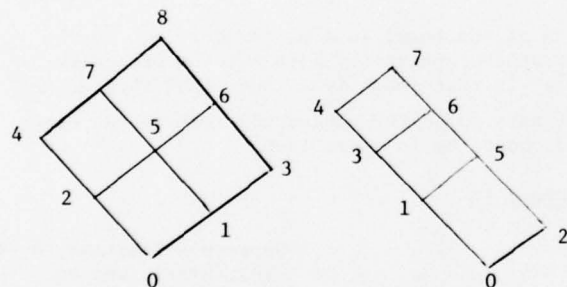
The sets of $P = \{e_1, \dots, e_s\}$ are called coordinate sets for the distributive lattice.

Corollary 9: For a fixed coordinate set $P = \{e_1, \dots, e_q\}$ for a distributive lattice the representation of every point b in the form of the theorem is unique if and only if $k = q^m$.

The proof of the corollary is straightforward and is omitted.

Note the distinction between the normal coordinates where a point is represented by the sum of, at most, m points from G , the bottom generating set, and the coordinates of a point for a coordinate set P which are just the multipliers which go with the atoms of the embedded Boolean algebra to give a representation of the point. We shall limit the discussion below to minimal coordinate sets, that is those containing q_1 points.

Example 3:



(i) A distributive lattice on $E(9)$ with $9=3 \cdot 3$. $A = \{3, 4\}$, $G = \{1, 2, 3, 4\}$. There are six possible minimal coordinate sets, namely $\{0, 5, 8\}$, $\{0, 6, 7\}$, $\{1, 2, 8\}$, $\{1, 4, 6\}$, $\{3, 4, 5\}$, and $\{2, 3, 7\}$. All the representations of the points are unique since $9 = 3^2$.

(ii) A distributive lattice on $E(8)$ with $8=4 \cdot 2$. $A = \{2, 4\}$, $G = \{1, 2, 3, 4\}$. There are 14 possible minimal coordinate sets, namely $\{0, 1, 3, 7\}$, $\{0, 1, 4, 6\}$, $\{0, 1, 6, 7\}$, $\{0, 3, 4, 5\}$, $\{0, 3, 5, 7\}$, $\{0, 4, 5, 6\}$, $\{0, 5, 6, 7\}$, $\{1, 2, 3, 4\}$, $\{1, 2, 3, 7\}$, $\{1, 2, 4, 6\}$, $\{1, 2, 6, 7\}$, $\{2, 3, 4, 5\}$, $\{2, 3, 5, 7\}$, and $\{2, 4, 5, 6\}$. The representations are not all unique, for example if $P = \{1, 2, 6, 7\}$ then $5 = 1 \cdot 4 + 2 \cdot 2 = 1 \cdot 4 + 6 \cdot 2 = 1 \cdot 4 + 7 \cdot 2$.

4. The Generalized Post Algebra.

We wish to ensure that our generalized Post algebra is functionally complete and this is achieved by using the characteristic functions defined on G .

Definition 4: A generalized Post algebra is represented by the octuple $\langle L, B, G, J, +, \cdot, 0, 1 \rangle$ where

- (a) $\langle L, +, \cdot, 0, 1 \rangle$ forms a complete distributive lattice.
- (b) B is the maximal set of complemented elements in L . It forms a Boolean algebra with complementation in B denoted by $'$.
- (c) G is the bottom generating set of L .
- (d) J is the set of characteristic functions defined for each element in G .

The inclusion of G explicitly in the generalized Post algebra is taken to mean that we assume exactly these constant functions. Although in some of the earlier theorems we may have appeared to be using the constant 0 this is always definable in the system by $a_i \cdot a_i'$ or $a_i \cdot a_j$ ($i \neq j$).

In the generalized Post algebra we have assumed the characteristic functions for just the points in G . It is necessary to show that this is sufficient.

Lemma 10: If $M = \{1, \dots, m\}$ are the atoms of a generalized Post algebra $\langle E(k), B, G, J, +, \cdot, 0, 1 \rangle$ then the characteristic function

$$J_0 x = \left\{ \sum_{j=1}^m J_j(j \cdot x) \right\}^1 = \prod_{j=1}^m J_j'(j \cdot x).$$

Proof: If $x = 0$ then $J_j(j \cdot x) = 0$ for all j ($1 \leq j \leq m$) and hence $J_0 x = 1$.

If $x \neq 0$ at least one $r_i \neq 0$. If $r_i \in \{i < \dots < a_i\}$ then $i \cdot x = i \cdot r_i = i$ and hence $J_i(i \cdot x) = 1$. Consequently $J_0 x = 0$ if $x \neq 0$.

Theorem 11: If $A = \{a_1, \dots, a_m\}$ is the set of atoms for the maximal Boolean algebra embedded in a generalized Post algebra $\langle E(k), \mathcal{B}, G, J, +, \cdot, 0, 1 \rangle$ then any characteristic function for a point $b \in E(k)$ can be expressed in the form:

$$J_b x = \prod_{i=1}^m J_{r_i} (a_i \cdot x)$$

where (r_1, \dots, r_m) are the normal coordinates of b .

Proof: (i) Suppose $x = b$.

$$b = \sum_{j=1}^m r_j \text{ so } a_i \cdot b = \sum_{j=1}^m a_i \cdot r_j = r_i.$$

$$\text{Hence } J_{r_i} (a_i \cdot b) = J_{r_i} r_i = 1.$$

(ii) If $x \neq b$ there exists at least one coordinate position in which x and b differ, say the j th position with coordinate x_j for x and r_j for b ($r_j \neq x_j$).

$$\text{Then } a_j \cdot x = \sum_{i=1}^m a_j \cdot x_i = a_j \cdot x_j = x_j.$$

$$\text{Hence } J_{r_j} (a_j \cdot x) = J_{r_j} x_j = 0 \text{ if } x_j \neq r_j \text{ so } J_b x = 0 \text{ if } b \neq x.$$

This completes the theorem.

Corollary 12: As an alternative statement of the same result is

$$J_b x = \prod_{i=1}^m [J(a_i \cdot x)]_{r_i}$$

$$\text{where } [J(a_i \cdot x)]_{r_i} = \begin{cases} J_{r_i} (a_i \cdot x) & \text{if } r_i = 0 \\ J_{r_i} (a_i \cdot x) & \text{if } r_i \neq 0. \end{cases}$$

The proof of the corollary is straightforward and is omitted.

We are now able to give the most important result, as below.

Theorem 13: A generalized Post algebra is complete.

Proof: The result is a direct consequence of the Representation theorem together with theorems 6 and 11. The $+$ and \cdot are respectively a sum-type and a product-type function. Theorem 6 shows that we can derive all the other constant functions in terms of the elements in G and Theorem 11 shows that all the other characteristic functions can be defined in terms of those in J . The result follows from the Representation theorem.

Our main interest, of course, is in the expression of an arbitrary switching function in the generalized Post algebra. Although we have the Representation theorem given earlier we can do rather better than this. Recall that if $k = q_1 \dots q_m$ the number of elements in G is

$$\text{given by } q = \sum_{i=1}^m (q_i - 1).$$

For convenience we label the elements of G $1, \dots, q$ in some manner. Note that the atoms of the distributive lattice are labelled $1, \dots, m$ so $G = \{1, \dots, m, \dots, q\}$.

Theorem 14: Any function $f_{x_1 \dots x_n}$ in a generalized Post algebra $\langle E(k), G, \mathcal{B}, J, +, \cdot, 0, 1 \rangle$ can be expressed in the form

$$f_{x_1 \dots x_n} = \sum_{j=1}^q j \cdot g_j x_1 \dots x_n$$

where $G = \{1, \dots, q\}$ and each g_j is a decisive function.

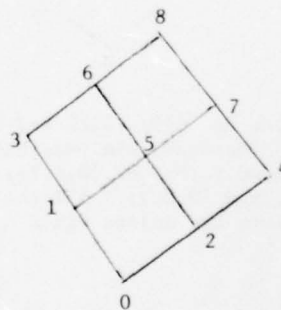
Proof: If we take the expression given in the Representation theorem and rearrange it using Theorem 6, an expression results of the type given in the theorem.

This result usually leads to a much more compact expression than the Representation theorem. There is further potential for simplification by using the result of Theorem 5. For this purpose it is convenient to rewrite the expression of Theorem 14 in terms of the chains from 0 to the atoms of the Boolean algebra, the expression being

$$f_{x_1 \dots x_n} = \sum_{i=1}^n \left(\sum_{j \in \{i < \dots < a_i\}} j \cdot g_j x_1 \dots x_n \right).$$

Each of the terms in a particular g_j may be introduced optionally into all the preceding g_j 's in that summation. Note that they do not all have to be introduced and that we can pick and choose as is convenient.

Example 4:



Suppose a function f on $E(9)$, $9=3 \cdot 3$, has an expression

$$f = \sum_{i=1}^8 i \cdot f_i.$$

The theorem 14 expansion is

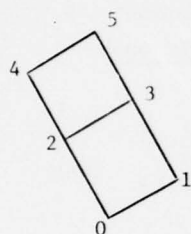
$$f = \sum_{j=1}^4 j \cdot g_j \text{ where}$$

the decisive functions g_j are given by

$$\begin{aligned}g_1 &= f_1 + f_5 + f_7 \\g_2 &= f_2 + f_5 + f_6 \\g_3 &= f_3 + f_6 + f_8 \\g_4 &= f_4 + f_7 + f_8\end{aligned}$$

The use of Theorem 5 says that all points where g_3 takes the value 8 are don't cares for g_1 and all the points where g_4 takes the value 8 are don't cares for g_2 .

Example 5: The sum digit for a 2-input base 6 adder, using + and · defined by the lattice shown.



The function f_{xy} is defined by the table shown. For such symmetric functions it is usually advantageous to make an initial transformation using the simple symmetric functions $w = x+y$, $z = x \cdot y$ and this leads to the transformed function f_{wz} . The -'s indicate don't cares.

f_{xy}	0	1	2	3	4	5	y	f_{wz}	0	1	2	3	4	5	z
0	0	1	2	3	4	5		0	0	-	-	-	-	-	
1	1	2	3	4	5	0		1	1	2	-	-	-	-	
x 2	2	3	4	5	0	1		w 2	2	-	4	-	-	-	
3	3	4	5	0	1	2		3	3	4	5	0	-	-	
4	4	5	0	1	2	3		4	4	-	0	-	2	-	
5	5	0	1	2	3	4		5	5	0	1	2	3	4	

We consider the alternative representations of f_{wz} .

From the Representation theorem

$$f_{wz} = \sum_{i=1}^5 i \cdot f_i w z$$

where

$$\begin{aligned}f_1 w z &= J_1 w \cdot J_0 z + J_5 w \cdot J_2 z \\f_2 w z &= J_1 w \cdot J_1 z + J_4 w \cdot J_4 z + J_2 w \cdot J_0 z + J_5 w \cdot J_3 z \\f_3 w z &= J_3 w \cdot J_0 z + J_5 w \cdot J_4 z \\f_4 w z &= J_2 w \cdot J_2 z + J_3 w \cdot J_1 z + J_4 w \cdot J_0 z + J_5 z \\f_5 w z &= J_3 w \cdot J_2 z + J_5 w \cdot J_0 z\end{aligned}$$

From Theorem 14 $f_{wz} = 1 \cdot g_1 w z + 2 \cdot g_2 w z + 4 \cdot g_4 w z$ where all the points for which g_4 takes the value 5 may be introduced as don't cares for g_2 (by Theorem 5). This gives the 3 tables shown below for these functions.

$g_1 w z$	0	1	2	3	4	5	z	$g_2 w z$	0	1	2	3	4	5	z
0	0	-	-	-	-	-		0	0	-	-	-	-	-	
1	5	0	-	-	-	-		1	0	5	-	-	-	-	
w 2	0	-	0	-	-	-		w 2	5	-	-	-	-	-	
3	5	0	5	0	-	-		3	5	-	-	0	-	-	
4	0	-	0	-	0	-		4	-	-	0	-	5	-	
5	5	0	5	0	5	0		5	-	0	0	5	5	-	

$g_4 w z$	0	1	2	3	4	5	z
0	0	-	-	-	-	-	
1	0	0	-	-	-	-	
w 2	0	-	5	-	-	-	
3	0	5	5	0	-	-	
4	5	-	0	-	0	-	
5	5	0	0	0	0	5	

giving

$$\begin{aligned}g_1 w z &= (J_1 w + J_3 w + J_5 w) \cdot (J_0 z + J_2 z + J_4 z) \\g_2 w z &= J_2 w + J_4 z + J_1 w \cdot J_1 z + J_3 w \cdot J_0 z + J_5 w \cdot J_3 z \\g_4 w z &= (J_5 z + J_0' z \cdot J_3' z (J_2 w + J_3 w) + J_0 z \cdot (J_4 w + J_5 w))\end{aligned}$$

There is clearly much greater potential for simplification by using theorems 14 and 5 than if we just limit ourselves to the Representation theorem.

5. Conclusion

The generalized Post algebra described in the earlier sections has the great advantage over the normal Post algebras in that it is complete. The only Post algebras that are complete are those with a maximal embedded Boolean algebra $B(2)$ and +, · being respectively the max and min functions. In the usual Post algebra as formalized by Epstein^{2,3} based on a chain of m fixed elements only k^m functions of n variables can be defined, as compared to k^n , the total number of n -place functions on $E(k)$.

References

- [1] J. Dussault, G. Metze and M. Krieger, "A multivalued switching algebra with Boolean properties", Proc. 1976 Int. Symp. on Multiple-Valued Logic, pp. 68-73.
- [2] G. Epstein, "The lattice theory of Post algebras", Trans. Amer. Math. Soc., Vol. 95, pp. 300-317.
- [3] G. Epstein and G. Horn, "Chain based lattices", Pac. J. Math., Vol. 55, 1974, pp. 65-84.

PREFILTERS OVER AN ARBITRARY BOOLEAN ALGEBRA

Ira Howard Sack

Kean College of New Jersey

ABSTRACT

We define the notion of a prefilter and then proceed to characterize the lattice of prefilters over an arbitrary Boolean Algebra. The lattice theoretic terminology can be found in Grätzer [5] and [6]. In the second section we further investigate the pseudo-prefilter consequence that was defined by Bloom []. We consider arbitrary prefilter matrices and prove a completeness theorem (Theorem 4 of Section II). Our logical terminology is that of Brown and Suszko [4].

SECTION I: ALGEBRA OF PREFILTERS

Brown [3] first defined the notion of the family of prefilters over a non-empty set X . According to his definition, a prefilter P over X is a family of subsets of X which has the following properties:

- (1) $X \in P$.
- (2) P is upward closed, i.e., if $X \in P$, $X \subseteq Y$, then $Y \in P$.
- (3) P has the finite intersection property (f.i.p., for short), i.e., the intersection of any finite number of non-empty subsets in P is non-empty.

Here we shall be concerned with examining the structure of the family of prefilters defined over an arbitrary Boolean algebra B . As it will be convenient to include B itself as a prefilter, we shall use the term "proper prefilter" in the sense Brown uses "prefilter".

Accordingly, we make the following definition of a proper prefilter:

Definition. A non-empty subset P of a Boolean algebra B is a proper prefilter iff:

- (1) P is upward closed, i.e., if $a \in P$ and $a \leq b$, then $b \in P$.
- (2) P has the finite intersection property, i.e., if $a_1, a_2, \dots, a_n \in P$, $n \geq 1$, then $a_1 \wedge a_2 \wedge \dots \wedge a_n \neq 0$ where 0 is the zero element of B .

Remark. The non-emptiness of P together with condition (1) imply that the unit of B is an element of P . Also condition (2), with $n = 1$, implies $0 \notin P$. Thus a proper prefilter of B is a proper subset of B .

We now expand the definition of prefilter to be:

Definition. P is a prefilter in B if either $P = B$ or P is a proper prefilter in B .

For a Boolean algebra B , $a \in B$, let \mathcal{J}_a denote the principal filter generated by a , where we recall that:

$$\mathcal{J}_a = \{b \in B \mid b \geq a\}.$$

Theorem 1. For any prefilter P , $P = \bigcup \{\mathcal{J}_a \mid a \in P\}$.

Proof. $a \in P$ implies $a \in \mathcal{J}_a$; so $P \subseteq \bigcup \{\mathcal{J}_a \mid a \in P\}$. Conversely, if $b \in \bigcup \mathcal{J}_a$, then $b \in \mathcal{J}_a$ for some $a \in P$,

i.e., $a \leq b$. Since $a \in P$ and P is upward closed, $b \in P$.

Theorem 2. Let $A \subseteq B$ be any subset with the f.i.p.. Then the set

$$P_A = U \{ \mathcal{J}_a | a \in A \}$$

is a proper prefilter.

Proof. Since A has the f.i.p., $0 \notin P_A$.

Henceforth, we refer to P_A as the prefilter generated by A . Let F_A denote the filter generated by A .

Lemma 1. F_A is a proper subset of B iff A has the f.i.p..

Theorem 3. P_A is proper iff F_A is proper.

Proof. Follows from Theorem 2 and Lemma 1.

The intersection of prefilters is a prefilter. More precisely, if the non-empty collection of $P_i, i \in I$ are prefilters of B , $\bigcap_{i \in I} P_i$ is a prefilter; it is seen to be the largest prefilter contained in every P_i . However, $\bigcup_{i \in I} P_i$ need not be a prefilter. We define $P_1 \vee P_2$ to be the least prefilter containing $P_1 \cup P_2$. Thus $P_1 \vee P_2$ is the intersection of all prefilters containing the set union of P_1 and P_2 . It exists since the "improper" prefilter B contains $P_1 \cup P_2$.

Similarly we define $\bigvee_{i \in I} P_i = P_A$ where $A = \bigcup_{i \in I} P_i$.

Theorem 4.

- (1) Let $A = P_1 \cup P_2$. Then $P_1 \vee P_2$ is proper if P_A has the f.i.p.;
- (2) $\bigvee_{i \in I} P_i$ is proper iff $\bigcup_{i \in I} P_i$ has the f.i.p.;
- (3) If $\bigvee_{i \in I} P_i$ is proper, then $\bigvee_{i \in I} P_i = \bigcup_{i \in I} P_i$.

Let $L^P = \langle L^P, \wedge, \vee \rangle$ denote the complete lattice of all prefilters of a Boolean algebra where

- (1) $P_1 \wedge P_2 = P_1 \cap P_2$;
- (2) $P_1 \vee P_2 = P_A$ where $A = P_1 \cup P_2$.

The following two definitions appear in Grätzer [5]:

Definition. An element a , of a complete lattice L is called compact if $a \leq \bigvee_{i \in I} x_i$ (where each $x_i \in L$) implies there exists a finite subset I_f of I such that $a \leq \bigvee_{i \in I_f} x_i$.

Definition. An algebraic lattice L is a complete lattice such that any $a \in L$ can be represented (compactly generated) as: $a = \bigvee \{ x | x \leq a, x \text{ compact} \}$.

Theorem 5. P is a compact prefilter in L^P iff $P = \mathcal{J}_{a_1} \cup \dots \cup \mathcal{J}_{a_n}$, n an integer ≥ 1 .

Proof. Suppose P is compact in L^P . If P is not proper, $P = \mathcal{J}_0$, the principal filter generated by the zero element of B ; otherwise, P is proper, so $P = \bigvee_{a \in P} \mathcal{J}_a = \bigcup_{a \in P} \mathcal{J}_a$. Hence by compactness, $P = \mathcal{J}_{a_1} \cup \dots \cup \mathcal{J}_{a_n}$, for some integer ≥ 1 .

Conversely, suppose

$$P = \mathcal{J}_{a_1} \cup \dots \cup \mathcal{J}_{a_n}.$$

If $P \leq \bigvee \{ G_i | i \in I \} = G$, then there are two cases to consider. First, if G is improper, $\bigcup G_i$ does not have the f.i.p., say $g_{i_j} \in G$ is such that

$$g_{i_1} \wedge \dots \wedge g_{i_n} = 0 \in B. \text{ Then}$$

$P \leq G_{i_1} \vee \dots \vee G_{i_n}$. Otherwise, $\bigvee G_i$ is proper, so $\bigvee G_i = \bigcup G_i$. Therefore, $a_j \in G_{i_j}, j = 1, \dots, n$ which implies $\mathcal{J}_{a_j} \leq G_{i_j}$. It follows that

$$\mathcal{J}_{a_1} \cup \dots \cup \mathcal{J}_{a_n} \leq G_{i_1} \vee \dots \vee G_{i_n};$$

so P is compact.

Corollary 5.1. L^P is a (compactly generated) algebraic lattice.

Proof. If P is a proper prefilter of L^P , then $P = U \{ \mathcal{J}_a \mid a \in P \}$; otherwise, $P = \mathcal{J}_0$. Thus L^P is compactly generated.

Theorem 6. If the collection of sets $P_i, i \in I$, is a chain in L^P , then $P = U_I P_i \in L^P$.

Proof. Suppose $U_I P_i$ does not have the f.i.p.. Then since the collection of P_i is a chain, some P_{i_0} is improper which implies $U_I P_i = B$, the unit element of L^P . Otherwise $U_I P_i = \bigvee_I P_i$ is a proper prefilter of L^P .

Theorem 7. Suppose B is a complete Boolean algebra and Q is a collection of prefilters $P_i, i \in I$ where all $P_i \subseteq B$. Then:

$$\bigwedge_I P_i = U \{ \mathcal{J}_{\bigvee_I a_i} \mid \text{for } i \in I, a_i \in P_i \} (*)$$

where the union on the right hand side of the equality is taken over the set of all principal filters generated by elements of B which are equal to the supremum of I elements of B where each of the I elements belongs to a different prefilter of Q .

Proof. Let S denote the right side of condition (*). Suppose $d \in \bigwedge_I P_i$. Then for each $i \in I$, $d \in P_i = U \{ \mathcal{J}_{a_i} \mid a_i \in P_i \}$; so for each i , $d \geq a_i$, some $a_i \in P_i$. Thus, $d \geq \bigvee \{ a_i \mid a_i \in P_i \}$ which implies $d \in S$.

On the other hand, suppose $d \in S$. Then d is an element of some principal filter $\mathcal{J}_{\bigvee_I a_i}$ where for $i \in I$, $a_i \in P_i \in Q$; hence, $d \geq \bigvee_I a_i$. So for and i , $d \in \mathcal{J}_{a_i} \subseteq P_i$ and thus $d \in \bigcap_I P_i$.

Corollary 7.1. Let Q be a collection of n prefilters P_1, P_2, \dots, P_n (all of the P_i being defined on a common Boolean algebra). Then,

$$P_1 \cap \dots \cap P_n = U \{ \mathcal{J}_{a_1 \vee \dots \vee a_n} \mid \text{for } 1 \leq i \leq n, a_i \in P_i \}$$

where the union on the right hand side of the above equality is taken over the set of all principal filters generated by elements which are formed as the supremum of n elements, each of the n elements belonging to a different prefilter of Q .

We now turn our attention to representation theorems for the lattice of filters (see Theorem 8) and the lattice of prefilters (see Theorem 11) of an arbitrary Boolean algebra. First we present some pertinent definitions for bounded lattices, i.e., lattices which contain a zero element and a unit element (which we denote as usual by 0 and 1).

Definition. An element c of a bounded lattice L is called clopen (or complemented) if there exists a c' (called the complement of c in L), $c' \in L$, such that:

$$c \vee c' = 1 \quad \text{and} \quad c \wedge c' = 0.$$

Definition. A bounded lattice L is zero-dimensional if it has a basis of clopen elements, that is, given any $c \in L$, c can be expressed as a join of clopen elements.

Lemma 2. If c and d are clopen elements of a bounded distributive lattice, then so are $c \wedge d$ and $c \vee d$.

Proof. The complement of $c \wedge d$ is $c' \vee d'$ since:

$$\begin{aligned} (1) \quad (c \wedge d) \vee (c' \vee d') &= \\ (c \vee c' \vee d') \wedge (d \vee c' \vee d') &= \\ = 1 \wedge 1 = 1. \end{aligned}$$

$$\begin{aligned}
 (2) \quad & (c \wedge d) \wedge (c' \vee d') = \\
 & (c \wedge d \wedge c') \vee (c \wedge d \wedge d') \\
 & = 0 \vee 0 = 0
 \end{aligned}$$

where we have applied the distributive laws in obtaining the above equalities. Similarly, with the aid of the distributive laws, we can show that the complement of $c \vee d$ is $c' \wedge d'$.

Corollary. The clopen elements of a bounded distributive lattice form a sublattice.

Lemma 3. For any Boolean algebra, the lattice of filters is isomorphic to the lattice of ideals.

Theorem 8. (due to S. Bloom). A complete lattice L is isomorphic to the lattice of filters of some Boolean algebra iff it is zero-dimensional, distributive, and has a compact unit element.

Proof. (necessity) For B , an arbitrary Boolean algebra, it is well-known that the lattice of filters of B forms a complete distributive lattice in which the principal filters constitute a clopen basis. To see that B (the unit filter) is compact, merely observe that if $B = \bigvee \mathcal{J}_i, i \in I$, then 0_B , the zero element of B , belongs to $\bigvee \mathcal{J}_i, i \in I$. Thus for some finite subset of I , say $\{j_1, \dots, j_n\}$, we have:

$$a_{j_1} \wedge \dots \wedge a_{j_n} = 0_B$$

where $a_{j_i} \in \mathcal{J}_{j_i}$ for $1 \leq i \leq n$.

Hence,

$$B = \mathcal{J}_{0_B} = \bigvee \mathcal{J}_{j_i}, i = 1, \dots, n.$$

(sufficiency) Suppose L is a complete distributive lattice with a clopen basis and 1 is a compact unit element of L . Designate the set of all clopen elements of L by F . By Lemma 2 and its corollary, F forms a comple-

mented sublattice of L . Since L is distributive so is F . Thus, F is a Boolean algebra in its own right. Define $f : L \rightarrow 2^F$ such that:

$$f(a) = I_a = \{x | x \in F, x \leq a\}.$$

We claim f is an isomorphism of L onto the lattice of ideals of F . It is easy to show f and f^{-1} are order-preserving mappings and that f is 1-1. Therefore we confine our proof to showing that f is an onto mapping.

Let I be any ideal of F . We claim: $I = I_a$ where $a = \bigvee \{y | y \in I\}$. Clearly $I \subseteq I_a$; we show $I_a \subseteq I$.

Suppose $x \in I_a$; then

$$x \leq a = \bigvee \{y | y \in I\}.$$

Since x is clopen and 1 is compact,

$$\begin{aligned}
 1 = x \vee x' & \leq \bigvee \{y | y \in I\} \vee x' \\
 & \leq \bigvee \{y | y \in I_f\} \vee x',
 \end{aligned}$$

for some finite subset I_f of I ; so,

$$\begin{aligned}
 1 \wedge x = x & \leq \bigvee \{y | y \in I_f\} \wedge x \\
 & \leq \bigvee \{y | y \in I_f\} \in I.
 \end{aligned}$$

Thus, $x \in I$ and $I_a \subseteq I$.

The theorem then follows from Lemma 3.

Remark. The following example shows that the lattice of prefilters of a Boolean algebra is generally not distributive.

Let B_8 be the Boolean algebra of all subsets of $\{1, 2, 3\}$. Choose pre-filters P_1, P_2 , and P_3 of B_8 :

$$P_1 = \mathcal{J}_{\{1\}}; P_2 = \mathcal{J}_{\{2, 3\}};$$

$$P_3 = \mathcal{J}_{\{1, 2\}} \cup \mathcal{J}_{\{1, 3\}}.$$

Then,

$$P_1 \wedge (P_2 \vee P_3) = P_1 \wedge B_8 = P_1;$$

however,

$$(P_1 \wedge P_2) \vee (P_1 \wedge P_3) \\ = \mathcal{J}_{\{1,2,3\}} \vee P_3 = P_3.$$

Definition. Let L be a lattice with 0. An element $a^* \in L$ is a (meet) pseudo-complement of $a \in L$ if:

$$a \wedge x = 0 \text{ iff } x \leq a^*,$$

for all $x \in L$.

Thus a^* is the maximum element of the set $\{x \in L \mid a \wedge x = 0\}$.

Definition. A pseudo-complemented lattice is a lattice in which every element has a pseudo-complement.

We recall some facts about pseudo-complemented lattices:

Lemma 4. Suppose $L = \langle L, \vee, \wedge, *, 0, 1 \rangle$ is a pseudo-complemented lattice, where $*$ denotes the unary operation of pseudo-complement. Then the elements of L satisfy the following ordering relations:

$$\begin{aligned} &\text{if } a \leq b, \text{ then } b^* \leq a^*; \\ &a \leq a^{**}; \\ &a^{***} = a^*; \\ &(a \vee b)^* = a^* \wedge b^*; \\ &a^* \vee b^* \leq (a \wedge b)^*; \\ &0^* = 1 \text{ and } 1^* = 0. \end{aligned}$$

Proof. See Grätzer [6].

Definition. If L is a pseudo-complemented lattice, then $a \in L$ is a closed element if $a^{**} = a$.

Theorem 9. Let L be a bounded pseudo-complemented lattice and suppose A denotes the set of all closed elements of L . Then A may be regarded as a Boolean algebra in which the meet operation is inherited from L ; the pseudo-complement operation of L serves as the complement operation in A and the join operation on A is defined for any $a, b \in A$ by:

$$a \vee_A b = (a^* \wedge b^*)^*.$$

Proof. See Theorem 4, page 58 in Grätzer [6].

Theorem 10. L^P , the lattice of pre-filters of a Boolean algebra B is pseudo-complemented.

Proof. Let 1_B denote the unit element of B . Then in L^P , $\{1_B\}$ is the zero element and B is the unit element. Clearly $\{1_B\}$ and B are mutually pseudo-complementary elements of L^P .

On the other hand suppose $Q = \bigvee \{\mathcal{J}_x \mid x \in X\} \neq \{1_B\}$ is a proper pre-filter and $Q \wedge R = \{1_B\}$, for some $R \in L^P$. Then R must be proper, hence representable as $R = \bigcup \{\mathcal{J}_y \mid y \in Y\}$. Let $S = \bigwedge \{\mathcal{J}_{\bar{x}} \mid \bar{x} \in X\}$ where \bar{x} denotes the Boolean complement of x in B . We claim $S = Q^*$. First note:

$$\begin{aligned} Q \wedge R &= \bigcup \{\mathcal{J}_x \mid x \in X\} \cap \bigcup \{\mathcal{J}_y \mid y \in Y\} \\ &= \bigcup_{X,Y} \mathcal{J}_x \vee y = \{1_B\}. \end{aligned}$$

Consequently $R \subseteq S$. But

$$Q \wedge S = \bigcup_X (\mathcal{J}_x \cap S) = \{1_B\};$$

so

$$S = \max\{R \in L^P \mid Q \wedge R = \{1_B\}\},$$

i.e., $S = Q^*$.

Corollary 10.1. The compact elements of L^P form a bounded pseudo-complemented sublattice.

Proof. Follows from Theorem 5, Corollary 7.1, and the fact that if

$$a = \bigvee_{i=1}^n \mathcal{J}_{x_i}$$

is a compact element of L^P , then $a^* = \mathcal{J}_{\bar{x}_1} \vee \dots \vee \mathcal{J}_{\bar{x}_n} = \mathcal{J}_{\bar{x}_1 \wedge \dots \wedge \bar{x}_n}$ is also compact. In particular, $(\mathcal{J}_x)^* = \mathcal{J}_{\bar{x}}$.

Corollary 10.2. If a is both closed and compact in L^P , then a is a principal filter.

Proof. The improper prefilter B is the principal filter generated by the zero element of B . Otherwise, if

$$a = \bigvee_{i=1}^n \mathcal{J}_{x_i}$$

is a proper filter which is compact in L^P , then $a^{**} = \mathcal{J}_{x_1} \wedge \dots \wedge \mathcal{J}_{x_n}$. If in addition a is closed, we may set

$$\bigcup_{i=1}^n \mathcal{J}_{x_i} = \mathcal{J}_{x_1} \wedge \dots \wedge \mathcal{J}_{x_n}.$$

Hence for some x_j , $1 \leq j \leq n$, we have $a = \mathcal{J}_{x_j}$.

Lemma 5. If $a, a_1, \dots, a_n \in L^P$ are closed and compact and

$$a \leq a_1 \vee \dots \vee a_n \neq B$$

where the join operation is in L^P , then $a \leq a_i$ for some i .

Proof. Follows from Corollary 10.2.

Below are summarized four facts that we have established about $L = L^P$:

- (1) L is an algebraic, pseudo-complemented lattice;
- (2) C , the compact elements of L , form a bounded pseudo-complemented sublattice of L ;
- (3) Every element of L is a join of elements in A , the Boolean algebra of closed elements of C ;
- (4) If $a \in A$ and all $s_i \in A$ satisfy:
 $a \leq a_1 \vee \dots \vee a_n \neq 1$
 (where the join operation is in L), then $a \leq a_i$, for some i .

We now prove a theorem suggested by S. Bloom which shows that any lattice L satisfying the four conditions listed above is isomorphic to the lattice of prefilters of some Boolean algebra. Thus conditions (1) through (4) characterize the lattice of prefilters L^P .

Theorem 11. If L is any lattice satisfying conditions (1) through (4) above, then L is isomorphic to the lattice of prefilters of the Boolean algebra dual to A via:

$$\varphi : x \in L \rightarrow \{a \in A \mid a \leq x\}.$$

Proof. First we shall show that φ is 1-1, order-preserving map.

Accordingly, if

$$\varphi(x) = \{a \in A \mid a \leq x\}$$

where $x \in L$, then in particular $\varphi(1) = A$. If however $x \neq 1$, then by condition (3) we have

$$x = \bigvee \{a \mid a \in \varphi(x)\}.$$

Furthermore, if $\varphi(x) = \varphi(y) \neq A$, then $x = \bigvee \{a \mid a \in \varphi(x)\} = \bigvee \{a \mid a \in \varphi(y)\} = y$. Thus it is always true that if $x \neq y$, then $\varphi(x) \neq \varphi(y)$. This establishes the fact that φ is a 1-1 mapping.

Clearly if $x > y$, then $\varphi(x) \supsetneq \varphi(y)$. But $\varphi(x) \neq \varphi(y)$, so $\varphi(x) \supsetneq \varphi(y)$, i.e., φ is order-preserving.

It remains to show that φ maps L onto the prefilters of A^d , the Boolean algebra dual to A . Now in A^d , every principal prefilter has the form $\{a \in A \mid a \leq_A a_i\}$, for some $a_i \in A$. But $\varphi(a_i) = \{a \in A \mid a \leq_A a_i\}$, hence it follows from Corollary 5.1 that every prefilter of A^d is either A or else proper and representable as

$$\bigcup_{i \in I} \varphi(a_i)$$

where all $a_i \in A$.

By conditions (2) and (4) we know if

$$x = \bigvee_{i \in I} a_i \neq 1$$

where all $a_i \in A$, then

$\varphi(x) \subseteq \{a \in A \mid a \leq a_i, \text{ for some } i\}$. But $\varphi(a_i) \subseteq \varphi(x)$, so

$$\varphi(x) = \bigcup_{i \in I} \varphi(a_i).$$

Since every proper prefilter of A^d is of the form

$$\bigcup_{i \in I} \varphi(a_i)$$

where all $a_i \in A$ and $\varphi(1) = A$, we conclude that φ maps L onto the lattice of prefilters of A^d .

SECTION II. PSEUDO-PREFILTER AND PREFILTER MATRIX CONSEQUENCES

A sentential language L is an absolutely free algebra of finite type generated by a countably infinite set of variables, $\text{var}(L)$, which is order isomorphic to the natural numbers. The i -th element of $\text{var}(L)$ is denoted x_i . Using the connectives of L , $\text{con}(L)$, as algebraic operations define on L , we may define an algebra of formulas (cf. Brown and Suszko [4]). Thus once $\text{con}(L)$ is specified both L and the algebra of formula of L are determined. Let the common language of classical two-valued and intuitionistic propositional logics be denoted by L_I where

$$\text{con}(L_I) = \{\wedge, \vee, -, \rightarrow, \leftrightarrow\}$$

and all connectives are binary except for $-$ which is unary.

We define C_P the "pseudo-prefilter" consequence operation on L_I by:

$\alpha \in C_P(X)$ iff α belongs to every pseudo-prefilter containing X , $X \subseteq L_I$

where F is a pseudo-prefilter iff $F = \text{nat}^{-1}(\bar{F})$ for \bar{F} some prefilter in \mathcal{B} , the free Boolean algebra over $\text{var}(L)$ and nat is the natural map of L_I onto \mathcal{B} . Let CL be the classical consequence operation on L_I (see Bloom [2]).

Remark. \mathcal{B} may be identified with equivalence classes of formulas of L_I where $\alpha \sim \beta$ if $CL(\alpha) = CL(\beta)$. Thus, $\text{nat}(\alpha) = \tilde{\alpha}$.

Theorem 1. F is a proper pseudo-filter (i.e., $F \neq L_I$) iff F is a non-empty subset of L_I satisfying:

- (1) $\beta \in F$ whenever $\alpha \in F$ and $\alpha \rightarrow \beta \in CL(\emptyset)$.
- (2) $\neg(\alpha_1 \wedge \dots \wedge \alpha_n) \notin CL(\emptyset)$ whenever $\alpha_1, \dots, \alpha_n \in F$, $n \geq 1$.

Proof. (necessity). If

$$\alpha \in F = \text{nat}^{-1}(\bar{F}),$$

where \bar{F} is a proper prefilter in \mathcal{B} , and $\alpha \rightarrow \beta$ is a tautology of CL , then $\tilde{\alpha} \rightarrow \tilde{\beta}$ is the unit element of \mathcal{B} . Hence $\tilde{\alpha} \leq \tilde{\beta}$ in \mathcal{B} ; so $\tilde{\beta} \in \bar{F}$ or equivalently, $\beta \in F$. Furthermore, if $\alpha_1, \dots, \alpha_n \in F$, for some $n \geq 1$, then $\neg(\alpha_1 \wedge \dots \wedge \alpha_n)$ is not a tautology; for otherwise, $\text{nat}(\alpha_1 \wedge \dots \wedge \alpha_n) = \tilde{\alpha}_1 \wedge \dots \wedge \tilde{\alpha}_n = 0$ in \mathcal{B} , which contradicts the fact that \bar{F} is a proper prefilter. Thus conditions (1) and (2) hold.

(sufficiency) Suppose $F \neq \emptyset$ satisfies conditions (1) and (2).
def.
It suffices to show that $\bar{F} = \text{nat}(F)$ is a proper prefilter in \mathcal{B} and that $F = \text{nat}^{-1}(\bar{F})$.

First note that if $\tilde{\beta} \in \bar{F}$, there is (by definition of \bar{F}) an $\alpha \in F$ such that $\tilde{\alpha} = \tilde{\beta}$. Hence, $\alpha \rightarrow \beta$ is a classical tautology and therefore, $\beta \in F$ follows from condition (1). Thus, $F = \text{nat}^{-1}(\bar{F})$.

We complete the proof by showing that \bar{F} is a proper prefilter in \mathcal{B} . Now $\bar{F} \neq L$; for otherwise, $\text{nat}(x_1 \wedge \neg x_1) \in \bar{F}$ which implies $x_1 \wedge \neg x_1 \in F$. Then $\neg(x_1 \wedge \neg x_1) \notin CL(\emptyset)$ follows from condition (2) which is absurd.

Next, suppose $\tilde{\alpha} \in \bar{F}$ and $\tilde{\alpha} \leq \tilde{\beta}$. Then $\text{nat}(\alpha \rightarrow \beta) = \text{nat}(\alpha \vee -\alpha)$, i.e., $\alpha \rightarrow \beta$ is a classical tautology. Thus, $\beta \in F$ or equivalently $\tilde{\beta} \in \bar{F}$, demonstrating that \bar{F} is upward closed. It remains to show that \bar{F} satisfies the f.i.p..

Suppose $\tilde{\alpha}_1, \dots, \tilde{\alpha}_n \in \bar{F}$ and $\tilde{\alpha}_1 \wedge \dots \wedge \tilde{\alpha}_n = \text{nat}(x_1 \wedge -x_1)$. Then $-(\alpha_1 \wedge \dots \wedge \alpha_n)$ is a classical tautology which contradicts condition (2). Thus \bar{F} is a proper prefilter in \mathcal{B} .

Let C_M designate the "prefilter" matrix consequence determined by the pair $M = \langle B, F \rangle$ where B is a Boolean algebra and F is an arbitrary prefilter in B ; thus:

$\alpha \in C_M(X)$, $X \subseteq L_I$, iff for all homomorphisms $h : L_I \rightarrow B$, $h(\alpha) \in T$ whenever $h(X) \subseteq T$.

Lemma 1. Let $M = \langle B, F \rangle$ be a prefilter matrix as described above. Let $h : L_I \rightarrow B$. Then $h^{-1}(F)$ is a pseudo-prefilter.

Proof. If $F = B$, then $h^{-1}(B) = L_I$, the improper pseudo-prefilter. Otherwise, suppose $F \neq B$, and $\alpha_1, \dots, \alpha_n \in h^{-1}(F)$, for some $n \geq 1$. If $-(\alpha_1 \wedge \dots \wedge \alpha_n)$ is a classical tautology, then $\alpha_1 \wedge \dots \wedge \alpha_n$ is a classical contradiction. But then, $h(\alpha_1 \wedge \dots \wedge \alpha_n) = h(\alpha_1) \wedge \dots \wedge h(\alpha_n) = 0$ in B , contradicting the properness of F . Thus $h^{-1}(F)$ satisfies condition (2) of Theorem 1. We complete the proof by showing condition (1) of Theorem 1 is satisfied, establishing that $h^{-1}(F)$ is a proper pseudo-prefilter. Accordingly, let $\alpha \in h^{-1}(F)$ and let $\alpha \rightarrow \beta$ be a classical tautology. We must show $\beta \in h^{-1}(F)$. Now,

$$h(\alpha \rightarrow \beta) = h(-\alpha) \vee h(\beta) = 1$$

in B ; so $h(\beta) \geq h(\alpha) \in F$. Thus,

$$\beta \in h^{-1}(F).$$

Remark. In the above proof we have made use of the well-known fact that if B is any Boolean algebra and h is any homomorphism such that $h : L_I \rightarrow B$, then $h(\alpha) = 1 \in B$ (i.e., $h(\alpha)$ is the unit element of B) whenever $\alpha \in CL(\emptyset)$.

Theorem 2. $C_P \leq C_M$, for any prefilter matrix consequence determined by $M = \langle B, F \rangle$.

Proof. Suppose $\alpha \in C_P(X)$, $X \subseteq L_I$, and $h : L_I \rightarrow B$ is any homomorphism such that $h(X) \subseteq F$. Then by Lemma 1, $h^{-1}(F)$ is a pseudo-prefilter containing X . So by definition of C_P , $\alpha \in h^{-1}(F)$. Hence, $h(\alpha) \in F$ which implies $\alpha \in C_M(X)$.

Thus, $C_P \leq C_M$.

Corollary 2.1. $C_P(X) \subseteq \bigcap C_M(X)$, any $X \subseteq L_I$.

Proof. Follows immediately from Theorem 2.

(Completeness) Theorem 3.

$C_P = \bigwedge_K C_M$, where K is the class of all matrices $M = \langle B, F \rangle$ where B is a Boolean algebra and F is a prefilter of B .

Proof. Suppose $\alpha \notin C_P(X)$, $X \subseteq L_I$. Then there is a pseudo-prefilter F in L_I such that $\alpha \notin F \supseteq X$; so, $\text{nat}(\alpha) \notin \text{nat}(F) \supseteq \text{nat}(X)$. Hence, $\alpha \notin C_M(X)$, where $M = \langle \mathcal{B}, \bar{F} \rangle$ is the logical matrix determined by \mathcal{B} , the free Boolean algebra over $\text{var}(L_I)$ and \bar{F} is that prefilter in \mathcal{B} such that $F = \text{nat}^{-1}(\bar{F})$.

Our next result is a recasting of the theorem proven in Sack [7].

Theorem 4. Let C_M be a prefilter matrix consequence where $M = \langle B, F \rangle$ where B is any Boolean algebra and F is a proper prefilter of finite cardinality n which is contained in B . Then

there exists an $\alpha \in L_I$ and an $X \subseteq L_I$ such that:

$$\alpha \in C_M(X) - C_P(X).$$

Proof. Let $X = \{x_1, \dots, x_{n+1}\}$ and let α denote the sentential formula

$$\bigvee_{1 \leq i < j \leq n+1} (x_i \wedge x_j).$$

Then if $h : L_I \rightarrow B$ is any homomorphism such that $h(X) \subseteq F$, h must identify two variables of X , i.e., $h(x_r) = h(x_s)$, for some $r \neq s$, where both $r, s \in 1, 2, \dots, n+1$. Then:

$$h(\alpha) = \bigvee (h(x_i) \wedge h(x_j)) \geq h(x_r) \in F.$$

Hence, $\alpha \in C_M(X)$.

Now let A be a set with $n+2$ elements, say $\{a_0, a_1, \dots, a_{n+1}\}$. Define the Boolean prefilter matrix: $N = \langle 2^A, T \rangle$ where T is all subsets of A which contain a_0 and have two or more elements.

Consider any homomorphism $h : L_I \rightarrow 2^A$ defined so that $h(x) = \{a_0, a_1\}$ for all $x_i \in X$. Then:

$$h(\alpha) = \bigvee (h(x_i) \wedge h(x_j)) = Va_0 = a_0 \notin T.$$

Thus, $\alpha \in C_N(X)$ and it follows from Theorem 3 that $\alpha \notin C_P(X)$.

Bloom [2] has observed that if B is a Boolean algebra and the matrix $M = \langle B, T \rangle$ where T is any subset of B such that $C_M \geq C_P$, then T is a prefilter in B . From this observation and the proof of Theorem 4, we obtain:

Corollary 4.1. For every matrix $M = \langle B, T \rangle$ where T is a finite proper subset of the Boolean algebra B , $C_M \neq C_P$.

Remark. C_P was originally defined in Bloom [2]; there it is proved that C_P is an example of a standard consequence which is not "finitely based",

i.e., C_P is a finite and structural consequence which is not definable by a finite number of (finite) structural rules of inference.

REFERENCES

- [1] Bloom, S. L., "A Representation Theorem for the Lattice of Standard Consequence Operations", Studia Logica, Vol. 34 (1975), p. 235-237.
- [2] Bloom, S. L., "Some Theorems on Structural Consequence Operations", Studia Logica, Vol. 34 (1975), p. 1-9.
- [3] Brown, D. J., "The Lattice of Prefilters", Notices AMS, Vol. 20 (1973).
- [4] Brown, D. J. and Suszko, R., "Abstract Logics", Dissertationes Mathematicae, CII (1973), p. 9-41.
- [5] Grätzer, G., Universal Algebra, Van Nostrand (1968).
- [6] Grätzer, G., Lattice Theory, W. H. Freeman (1971).
- [7] Sack, I. H., "Prefilter Consequence Operators and their Realization by Logical Matrices with Multi-valued Truth Sets", Reports on Mathematical Logic, No. 5 (1975).

AUTOMATED GENERATION OF MODELS AND COUNTEREXAMPLES AND
ITS APPLICATION TO OPEN QUESTIONS IN TERNARY BOOLEAN ALGEBRA *

Steve Winker
Research Associate
Northern Illinois University
DeKalb, IL 60115

L. Wos
Applied Mathematics Division
Argonne National Laboratory
Argonne, IL 60439

Abstract

The thrust of this paper is: first, to answer certain previously unanswered questions in the field of Ternary Boolean algebra; second, to describe the method, utilizing an automated theorem-proving program as an invaluable aid, by which these answers were obtained; and third, to informally give the characteristics of those problems to which the method can be successfully applied. The approach under study begins with known facts in the form of axioms and lemmas of the field being investigated, finds by means of certain specified inference rules new facts, and continues to reason from the expanding set of facts until the problem at hand is solved or the procedure is interrupted. The solution often takes the form of a finite model or of a counterexample to the underlying conjecture. The model and/or counterexample is generated with the aid of an already existing automated theorem-proving procedure and without any recourse to any additional programming.

1. Introduction

In this paper we give a procedure which in part complements the ongoing effort of using computers to find mathematical proofs. The procedure is used to automatically generate finite models of various mathematical theories and/or finite counterexamples to conjectures (for more details refer to a paper submitted for publication⁸). The ability to thus generate models and counterexamples when combined with an automated theorem-proving program provides one with the more effective attack on certain open questions in mathematics and logic. We invite the submission for consideration of such open questions as those answered in section 2 and of those informally classified in section 4. Such new problems are of value in the development, extension, and refinement of the approach under discussion, and there is, of course, the carrot that some of them may be thus solved. This in fact was the case for those problems in the field of Ternary Boolean algebra which were considered and treated with the procedure under investigation.

The organization of the paper is as follows. In section 2, Ternary Boolean algebras are defined, the previously open questions are stated, and answers are given including certain key models and

lemmas. In section 3, the procedure for generating models is discussed. It is here we give both the language in which the problems must be stated and the underlying inference rules which generate additional facts and permit validation of the models. Finally, in section 4 we discuss the kinds of problems and fields of mathematics which are most amenable to the approach of section 3.

We conclude the section with the following summary. The proofs of the various lemmas and the models and counterexamples given here were obtained with the aid of the computer and, more specifically, with extensive use of an automated theorem-proving program^{5,6,9,10}. No additional programming was required to obtain our results, nor would any be necessary to attack similar open questions. Since, in addition to model generation, the approach herein can be used for deductive reasoning, it may be of use in areas other than mathematics.

2. Ternary Boolean Algebras: The Solution to Some Open Questions

A Ternary Boolean algebra⁴ is a set S together with a Ternary function f and a unary function g which, for all elements in S , satisfy:

- (1) $f(f(v,w,x),y,f(v,w,z)) = f(v,w,f(x,y,z))$
- (2) $f(y,x,x) = x$
- (3) $f(x,y,g(y)) = x$
- (4) $f(x,x,y) = x$ and
- (5) $f(g(y),y,x) = x$.

One natural question to be asked is: Of the five axioms defining a Ternary Boolean algebra, which (if any) among them are dependent on the remaining? Chinthayamma³ announced in 1969 without proof that axioms 4 and 5 were dependent on the subset consisting of 1, 2, and 3. This left open the question of which of the remaining proper subsets of the five axiom set are strong enough to define a Ternary Boolean algebra. Then, for each subset, U , which is too weak, one can ask: What are the more interesting properties of U , and what is the cardinality of the smallest model which satisfies U and simultaneously fails to satisfy the remaining of the five axioms which define a Ternary Boolean algebra? The answer to the foregoing can be thus summarized.

The most important facts, obtained with the procedure discussed in section 3, are the following. Let T be the set of the five defining

* Work performed under the auspices of the U.S. Department of Energy and NSF grant MCS77-02703.

axioms, U_1 be obtained from T by the omission of axiom 1, U_2 by the omission of axiom 2, and U_3 by the omission of axiom 3. There are models, exhibited later in this section, showing that each of U_1 , U_2 , and U_3 is insufficient to define a Ternary Boolean algebra. Equivalently, axioms 1, 2, and 3 are each independent of the remaining four axioms in T . Next, one can show that in U_2 , $f(g(g(x)), x, y) = g(g(x))$ for all x and y . In U_3 , $f(x, g(x), y) = y$ for all x and y . Axioms 1 and 4 together imply that $f(x, y, x) = x$. In T , $g(g(x)) = x$ and finally, from axioms 4 and 5 one can show that, if there exists an element a in the set under consideration such that $g(a) = a$, then the set consists of just a . Since the establishment of assertions of the type just given reduces the work required to generate the appropriate models, we turn to their proof.

The last of the assertions can be seen from $x = f(g(a), a, x) = f(a, a, x) = a$ for any x . That $f(x, y, x) = x$ in the presence of 1 and 4 can be seen by simply setting $v = w$ in 1.

Next, we see that $g(g(x)) = x$ when all of T is present by setting in axiom 1, $w = x$, $v = g(g(x))$, $y = z = g(x)$, and applying 2 and 3. (It should be noted that the program employed by the procedure section 3 does not find this and similar proofs by considering all or various substitutions into the various axioms. Instead, selections from the axioms and derived facts are made and presented to the inference rules which then in turn make a deduction with an emphasis on generality. Then a subprocedure is invoked which compares the "new" result with those already retained. The procedure will purge, for example, a deduction D' in favor of D , regardless of which is newer, when D is more general than D' .)

Returning to the assertions to be proved, we have U_2 implies $f(g(g(x)), x, y) = g(g(x))$. In axiom 1, just set $x = y = g(w)$, and $v = g(g(w))$, and apply 3, 4, and 5 to get an alphabetic variant of the desired result.

Finally, the following six-step proof shows that one can prove from U_3 , $f(x, g(x), y) = y$.

Proof. $f(f(v, g(v), x), x, v) = f(v, g(v), x)$ from the substitution into axiom 1, respectively for v, w, x, y, z of $v, g(v), x, x, v$, and from applications of $f(x, y, x) = x$ and axiom 4. $f(f(v, g(v), x), v, x) = f(v, g(v), x)$ from the substitution into axiom 1 respectively of $f(v, g(v), x), x, v, v, x$, and applications of the previous step and of axiom 2 and axiom 4. $f(f(v, w, g(w)), w, v) = v$ from substitution into axiom 1 of $v, w, g(w), w, v$, and applications of $f(x, y, x) = x$ and axiom 5. $f(w, g(w), f(v, w, g(w))) = f(v, w, g(w))$ from the substitution into axiom 1 of $v, w, w, g(w), g(w)$, and applications of axiom 2. $f(f(v, g(v), x), x, f(x, v, g(v))) = f(v, g(v), x)$ from the substitution into axiom 1 of $v, g(v), x, x, f(x, v, g(v))$, and applications of axiom 4 and of the previous step. Finally, $f(v, g(v), w) = w$ from the substitution into axiom 1 of $f(v, g(v), w), w, f(w, v, g(v)), v, w$, and applications of axiom 2 and steps 3, 5, and 2, which completes the proof.

Thus we have completed the proofs of the various assertions given above, and we can now turn to the direct consideration of the question of axiom dependence for Ternary Boolean algebras. As stated earlier, the facts are that axioms 4 and 5 are dependent on the remaining three, but none of axioms 1, 2, or 3 is dependent on the other four. Equivalently, the only subsets of the given set of five axioms defining a Ternary Boolean algebra which are strong enough for the definition are those subsets which contain at least axioms 1, 2, and 3.

First, to see that axiom 1 is independent of the others, merely consider the three-element model, consisting of a, b , and c , with $g(g(g(x))) = x$, $g(a) = b$, $g(b) = c$, $f(a, b, a) = b$, $f(b, c, b) = c$, $f(c, a, c) = a$, and all of the triples satisfying axioms 2 through 5. The substitution into axiom 1 respectively for v, w, x, y, z of a, b, c, c, a respectively yields $a = b$, which shows that axiom 1 does not hold.

Next we establish the independence of axiom 2. Consider the model consisting of the three elements, a, b , and c such that $g(g(g(x))) = x$ for all x , $g(x)$ not equal x for all x , $g(a) = b$, $f(x, x, y) = x$ for all x and y , $f(g(y), y, x) = x$ for all x and y , and $f(g(g(x)), x, y) = g(g(x))$ for all x and y . First note that $g(b) = c$ and $g(c) = a$. Then, by a tedious examination of all triples, one can verify that axioms 1, 3, 4, and 5 hold. The violations of axiom 2 are three instances of $f(g(g(x)), x, y) = g(g(x))$, namely, $f(a, b, b) = a$ and $f(b, c, c) = b$ and $f(c, a, a) = c$.

(Before turning to the final dependency question and then to the question of minimal counterexamples, we make the following observations which are important to the understanding of the procedure by which the results were obtained.) The procedure of section 3 does the tedious checking of the various triples required to validate the proposed model. One phase of the process is the check for consistency. For example, in the just-given model one could evaluate $f(a, a, b)$ with axiom 4 or with axiom 3. In both choices the same value must be obtained. Also note that the model is not presented to the procedure by means of precisely those equalities given in the previous paragraph, but rather by a set which includes the axioms to be satisfied. The other equalities are derived and used to narrow the search for the desired model or counterexample.

Now to establish the independence of axiom 3, just replace in the previous three-element model the requirement of $f(g(g(x)), x, y) = g(g(x))$ with the requirement of $f(x, g(x), y) = y$. Axiom 2 will now hold since, for example, $f(a, b, b) = b$ by the new equality. On the other hand, axiom 3 is now violated for the value of three triples has changed. The three violations are $f(a, b, c) = c$, $f(b, c, a) = a$, and $f(c, a, b) = b$ from the new equality.

(For the interest of the reader, axioms 4 and 5 may be proved from axioms 1, 2, and 3 as follows.

The proof that $g(g(x)) = x$ depends only on axioms 1, 2, and 3. Substitute $g(g(x)), x, y, g(x), g(x)$ in axiom 1 and apply 2, 3, and $g(g(x)) = x$ to yield 4. Substitute $x, g(y), g(y), g(g(y)), g(g(y))$ in 1 and apply 2, 3, and $g(g(x)) = x$ to yield 5.)

The final question, that of minimality for the counterexamples to axiom dependency, can be settled with the following argument. For two-element models, either there exists an x with $g(x) = x$, or g interchanges a and b . The first possibility is eliminated by an earlier remark, i.e., the presence of axioms 4 and 5 would force then $a = b$ to be true. For the second possibility, axiom 5 forces $f(b, a, a) = a$ and $f(a, b, b) = b$, which says that axiom 2 holds. Also in this case, axiom 5 forces $f(a, b, a) = a$ and $f(b, a, b) = b$, while axiom 4 forces $f(a, a, b) = a$ and $f(b, b, a) = b$, which says that axiom 3 holds. A similar analysis shows that axiom 1 also holds in this case. So the smallest counterexample to the dependence of 1, 2, or 3 consists of three elements.

Returning to the discussion of the models themselves, we illustrate a somewhat different use of the procedure of section 3. The problem for consideration is the generation, if such exists, of an asymmetric three-element model of U_2 . The specific objective is that of determining whether or not any three-element models exist satisfying axioms 1, 3, 4, and 5, violating axiom 2, and with g not an onto mapping. There are two such models. In each we have $g(g(g(x))) = g(x)$, $g(a) = b$, $g(b) = c$, $g(c) = b$, $f(x, x, y) = x$, $f(g(y), y, x) = x$, and $f(x, y, g(y)) = x$. In both, from the earlier results, $f(x, y, x) = x$ and $f(g(g(x)), x, y) = g(g(x))$; also $f(a, c, c) = a$ (substitute a, c, b, a, c in axiom 1 and apply 3, 4, and 5). They differ in that in one, $f(a, b, b) = b$ while in the other $f(a, b, b) = a$. In both the violations of axiom 2 are $f(c, a, a) = c$ and $f(a, c, c) = a$. Examination of the various substitutions shows that axiom 1 holds for both models.

When the problem for the previous paragraph is replaced by the corresponding one for U_3 , we find that there is but one asymmetric model of 1, 2, 4, and 5 violating axiom 3. The model is quite like that which was just given except that axiom 3 is replaced by axiom 2, $f(g(g(x)), x, y) = g(g(x))$ is replaced by $f(x, g(x), y) = y$, $f(a, c, b) = b$, and $f(c, a, b) = b$. The last two equalities are, of course, the violations of axiom 3.

The above models are the only three-element models of U_2 and U_3 violating axioms 2 and 3, respectively. First, observe that there are only two possibilities for the effect of g on three elements: the "symmetric" and the "asymmetric" possibilities given above ($g(x)$ cannot be x for any x as noted previously). Secondly, the possibilities for values of f not given above are eliminated by contradictions to axiom 1 which were found by use of the program.

3. The Main Procedure for Generating Models

Since the presentation in subsections 3.1 to 3.6 is brief and somewhat intuitive, we have

included an example in 3.7 to aid one's understanding of the basic procedure and of the various underlying concepts. The example illustrates the iterative nature of our procedure. Its development in part parallels the mathematical treatment of section 2, and illustrates the method employed to obtain one of the asymmetric models of U_2 given there.

3.1 Overview

The main objective is the development and implementation of a more complete procedure for attacking open questions in mathematics and logic. It is important, in the treatment of such questions, to have a procedure for generating models and counterexamples. Such a procedure, based on an existing automated theorem-proving program, is the focus of attention in this section. For the side of the problem concerned with finding proofs for "true" theorems, there exists computer programs in various stages of development whose objective is that of proof finding. The proofs so obtained are usually ones by contradiction. In general, one begins with a set of statements, some of which correspond to axioms and lemmas while others correspond to the denial of the theorem to be proved, and continually applies rules of inference until the unsatisfiability of the set of statements is established. What is missing from such programs is an automated treatment of the other side of the problem, namely, the establishment with the aid of the computer that a desired result does not hold. Put differently, when the given set of statements is satisfiable, one would like to have a procedure which establishes that fact -- a procedure which generates a counterexample to the purported theorem. Although the fundamental theorems for the first-order predicate calculus prevent us from having a decision procedure, we have been able to establish that certain results are non-theorem and thus answer certain previously open questions as those of section 2.

With the object of automatedly generating models, we turn to a brief description of the various components.

3.2 The Language

The chosen language, employing the clause representation,² is one which is closely related to the extended first-order predicate calculus.² The equality symbol (predicate) is thus treated as a special symbol with its meaning "built-in". All variables and all given and inferred statements are (implicitly) assumed to be universally quantified. The existentials have been replaced by skolem functions,² functions of the appropriate universally quantified variables on which the existentials depend. The ensemble of statements is (implicitly) assumed to be in conjunctive normal form -- there is an implicit "and" between pairs of statements, and implicit "or" between the elements of each statement.

3.3 The Inference Rules

Although the procedure has access to a number of inference rules, the two most frequently employed are respectively generalizations of equality substitution and of modus ponens. The first rule, called paramodulation,⁹ takes clauses (statements) in pairs and attempts to substitute from one into the other. A substitution occurs if and only if a common domain of definition can be found for one of the arguments of the "from" clause and for a term in the "into" clause. The "from" clause must be the correspondent of some given statement of equality, while no essential restriction is placed on the term into which the substitution takes place. For further clarification and also to see that we are employing a generalization of equality substitution, note that paramodulation in one inference step takes the pair of statements (clauses), finds when possible a most general replacement of the variables (universal instantiation) in both which will permit a straightforward application of equality substitution, and applies an equality substitution rule to the instantiated pair.

As for the other main rule, called hyper-resolution,⁷ the following cursory description may suffice. The rule takes a set, Q_1, Q_2, \dots, Q_n , of positive assertions, an "if-then" statement of the form $Q'_1 \& Q'_2 \& \dots \& Q'_N \rightarrow R$, finds (where possible) a most general variable replacement to simultaneously apply to the Q_i and Q'_i which permits a modus ponens type inference, and then makes that inference from the thus instantiated clauses. There is the additional requirement that R be positive.

3.4 Usage

No programming is required of the individual who intends to use the procedure as an aid in answering questions. What is required is the preparation of the problem in one of two forms. The procedure itself accepts problems in the clause forms discussed in 3.2. Recall that one is therefore using a conjunctive normal form in which the variables that appear are implicitly universally quantified, the existentials have been replaced by appropriate skolem functions, and an implicit "and" occurs between clauses while an implicit "or" occurs between the literals of a clause. For example, the clause equivalent of that axiom which states that the nonzero elements of a field possess a multiplicative inverse consists of the two clauses, $\text{EQUAL}(X, 0) \text{ EQUAL}(F(H(X), X), 1)$ and $\text{EQUAL}(X, 0) \text{ EQUAL}(F(X, H(X)), 1)$, where F denotes product and H inverse and 1 the multiplicative identity. (There are other valid clause encodings of this axiom.) The scope of universal quantification is just that single clause, i.e., an occurrence of the variable "X" in two clauses does not mean the "same" variable. So one can submit the problem directly to the main procedure by encoding it in clause form.

On the other hand, if one finds such an encoding difficult or inconvenient, one can instead choose to represent the problem in the first-order predicate calculus. There is no requirement, in such a choice, of using prenex form and no

restriction on the use of the various boolean connectives. The availability to the user of the first-order representation is due to the existence of a system, called TAMPR,¹ designed and implemented at Argonne National Laboratory. TAMPR, although designed for program transformations of a different type, will take the problem in its first-order form and produce the clauses with appropriate replacement of existentials by functions.

3.5 The Procedure Itself

We choose to slant our description toward the informal and intuitive. The procedure can be said to be divided into a number of phases from which the user can choose any or all. There is the lemma generation or fact finding, the counterexample and/or model building, the validation of the computed counterexample and/or model, the rejection of such, and the proof of the theorem. Thus, despite the bias that may exist when considering a particular open question, the procedure may succeed in proving the corresponding purported theorem or may instead generate a counterexample. It is the second of these alternatives (in the closely related area of generating models for consistent axiom systems) on which we concentrate.

The approach is at present one of iteration in which one begins with a set of statements which may include axioms, lemmas, and various conjectures about the definition or structure of the sought-after counterexample or model. (Throughout the rest of this subsection we make no distinction between "counterexample" and "model.") One then makes a series of computer runs with the intention of appropriately adding to and/or deleting from the original input set. Additions are either lemmas which were not already present or "promising" extensions to the definition of a model. The validity of any of the former can be established by examining its proof tree to show that only already-proven theorems are relied upon, while validity of various of the latter may remain in question until the model is completed because of reliance on other conjectured conditions. (The derivation information is included in the computer output of each run.) Deletions, on the other hand, are usually the result of detecting inconsistency in the conditions defining the model. Inconsistency is signalled by the deduction of "contradiction" by the program. Such deletions often cause a fair amount of backtracking because of the corresponding necessity of making new conjectures about the structure of the model, and this may in turn require duplication of earlier runs but with, of course, the new conjectures. Thus the additions and deletions made by the user in earlier runs in part determine the nature of later runs.

One of the nice features of our procedure is that these computer experiments are accomplished with the aid of an existing automated theorem-proving program and require no additional programming. Each experiment or run is terminated either by exceeding memory or time, or by having made all possible new inferences, or deducing

contradiction. The second termination condition occurs either when the model has been both successfully completed and validated or when the model is partially specified but no inconsistencies exist therein. One can differentiate between the two cases by simply examining the computer output. The third termination condition signals model inconsistency or proof of the theorem thus answering the open question under study. Examination of the proof is sufficient to determine which is the case. Finally, the first termination condition can occur with any use in any phase of the procedure.

For each of the phases of the procedure (listed at the beginning of this subsection), the following remarks in general hold. Lemmas are found through use of the inference rule, paramodulation, which is a generalization of equality substitution (see 3.3). New information is checked against that already present, and the more general fact is retained and the less general purged. Both the building and validation of the model are accomplished through application of hyper-resolution, a modus ponens-like rule discussed also in 3.3. The rejection of the model being developed is through a combination of inferences from paramodulation and from hyper-resolution. And finally, the proofs of theorems may be obtained from various inference rules, but paramodulation is often the most successful.

3.6 Applications

The procedure under investigation has been used to generate the multiplication table for various semi groups, for successfully searching for models establishing the correctness of certain conjectures, to generate counterexamples and thereby refute various possible axiom dependencies as in section 2, and for finding proofs for various theorems as also given in section 2. All models and axiom sets considered so far have been of small cardinality.

3.7 An Example of Our Procedure

Some of the automated theorem prover runs made in searching for one model of U2 are listed below to indicate the degree of our reliance on the computer. As might be expected, the search includes tests which appear inconclusive or unnecessary in retrospect.

1. Paramodulation runs proved TBA axioms 4 and 5 from axioms 1, 2, and 3, and incidentally derived $g(g(x)) = x$ from axioms 1, 2, and 3.
2. A paramodulation run attempting to prove axiom 2 from axioms 1, 3, 4, and 5 proved neither axiom 2 nor $g(g(x)) = x$. Incidentally, $f(x,y,x) = x$ was derived.

The failure to prove axiom 2 motivated the search for a counterexample. The fact that $g(g(x)) = x$ was not derived suggested that a model violating $g(g(x)) = x$ be attempted. Such a model would necessarily violate axiom 2. It could be based on one generator (an a for which $g(g(a)) \neq a$) rather than two (an a

and b for which $f(b,a,a) \neq a$), possibly requiring fewer elements, fewer defining relations, and less computer time for verification.

3. Paramodulation run seeking consequences of axioms 1, 3, 4, and 5, in conjunction with $g(g(g(x))) = g(x)$, $g(f(x,y,z)) = f(g(x),g(y),g(z))$, and $f(x,y,z) = f(x,z,y)$. Axiom 2 was not proved, but the last equality with axioms 3 and 5 yielded $g(g(x)) = x$.

Because a model with $g(g(a)) \neq a$ was being sought, the last equality was not used for subsequent models. The possibility that $g(g(g(x))) = g(x)$ might imply $g(g(x)) = x$ was not tested at this time.

4. A paramodulation run deriving consequences of axioms 1, 3, 4, and 5, in conjunction with $g(f(x,y,z)) = f(g(x),g(y),g(z))$ and $f(a,x,g(g(a))) = a$, derived no undesirable consequences.

The latter equality was suggested by an examination of the proof of $g(g(a)) = a$ from axioms 1, 2, and 3. This proof used the instance of axiom 2, $f(a,g(g(a)),g(g(a))) = g(g(a))$; if this term had the value a instead, $g(g(a)) = a$ would not be proven. $f(a,x,g(g(a))) = a$ generalizes the second argument of f .

5. Partial-model run in which values for $f(a,c,c)$, $f(c,a,a)$, and $f(a,b,b)$ had not yet been determined. (Here b and c refer not to generators but to $g(a)$ and $g(g(a))$.)
6. Partial-model run in which the value for $f(a,c,c)$ had not yet been determined.
7. Model validation run verifying the first asymmetric model of section 2.

4. Requirements for New Problems

The submission of problems for consideration by the procedure described in this paper is most welcome. To be admissible, such problems must be representable in the first-order predicate calculus. If the object is the generation of a counterexample or model, there is the assumption that one of small finite cardinality will suffice. On the other hand, if the object is the finding of a proof for a purported theorem, we only require a statement of the theorem and a set of axioms characterizing the field from which the theorem is taken. In addition we find value in having a list of the important lemmas of the field.

Among those areas whose open questions may be most amenable to attack with the automated procedure are the theory of semi groups, elementary group theory, ring theory, the theory of Ternary Boolean algebras, Boolean algebra, and Tarskian geometry. For the type of question, one might consider, for example, questions concerned with the equivalence of axiom systems, questions about the existence of certain mappings, and questions

of axiom dependency. On the other hand, phrases such as "for all integers n " and "for all functions f " strongly suggest the lack of an appropriate mechanism to handle the question.

We conclude by remarking that consideration of open questions and problems of the type just discussed should, when subjected to treatment by computer programs of the type underlying this paper, lead to the alternation of the solution of some problems followed by the development of more successful automated procedures followed by the solution to others....

References

1. Boyle, J. and Matz, M., Automating Multiple Program Realization, *Proceedings of the Symposium on Computer Software Engineering*, Polytechnic Press, New York, 1976, pp. 421-456.
2. Chang, C. L. and Lee, R. C. T., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.
3. Chinthayamma, Sets of independent axioms for a ternary Boolean algebra (preliminary report), *Notices of the Amer. Math. Soc.*, Vol. 16, No. 4, June 1969, p. 654.
4. Grau, A. A., Ternary Boolean Algebra, *Bulletin of Amer. Math. Soc.*, Vol. 53, No. 6, June 1947, pp. 567-572.
5. McCharen, J. D., Overbeek, R. A., and Wos, L., Problems and Experiments for and with Automated Theorem-Proving Programs, *IEEE Transactions on Computers*, Vol. C-25, No. 8, August 1976, pp. 773-782.
6. Overbeek, R. A., An implementation of hyper-resolution, *Comput. Math. Appl.*, Vol. 1, 1975, pp. 201-214.
7. Robinson, J. A., Automatic Deduction with Hyper-resolution, *International Journal of Computer Math.*, Vol. 1, 1965, pp. 227-234.
8. Winker, Steve, Generation and Validation of Finite Models and Counterexamples using an Automated Theorem Prover, submitted for publication.
9. Wos, L. and Robinson, G. A., Paramodulation and Set of Support, *Proc. IRIA Symposium on Automatic Demonstration*, Versailles, France (1968), Springer-Verlag, 1970, pp. 276-310.
10. Wos, L., Robinson, G. A., Carson, D. F., and Shalla, L., The concept of demodulation in Theorem proving, *J. Assn. Comput. Mach.*, Vol. 14, 1967, pp. 698-709.

A SUMMARY OF INVESTIGATIONS INTO THREE AND FOUR VALUED LOGICS

George Epstein

Indiana University
Bloomington, Indiana 47401

References

There are a number of basic 3 and 4-valued relationships which have been neglected in the literature. Some of these are relationships which involve key connectives such as strong negation. The latter operation has logical properties corresponding to electrical properties of inverters which are used today in the design of all binary computers (and also used indirectly through the use of NAND or NOR circuits, at the feedback paths within flip-flop circuits, and at the inner inversion points of certain cascaded amplifiers). The reason for such neglect seems to be two-fold: (1) The technical development for general n camouflages specifics for certain low values, such as $n=2, 3$, or 4 ; and (2) The concomitant avoidance of immediate practical applications, encourages neglect of practical relationships which are needed by the computer scientist or engineer.

The theoretical background for this work occurs as early as 1941, with the algebras of G. Moisil. The inability of these algebras to model Łukasiewicz logics with more than 4 values was noted in 1968 by A. Rose. There are similar results for the P-algebras of Epstein and Horn with either the Meskhi or Cignoli conditions, as reported in the Bulletin of the Section of Logic, Polish Academy of Sciences, 6, #4, pp.156-160, 1977.

The practical reason for focus on the 3 and 4-valued cases is the continuing interest in 3-valued digital circuitry and computers, evidenced in our 8 annual symposia on multiple-valued logic of this decade, and the most recent application to 4-valued digital circuitry by Signetics Corporation, reported in May, 1977 at the Seventh International Symposium on Multiple-valued Logic at Charlotte, North Carolina. This encourages a fresh look at interconnections and relationships which hold for the 3 and 4-valued cases, with special emphasis on the key connectives which appear in practical applications.

1. Cignoli, R., The lattice of global sections of sheaves of chains over Boolean spaces, Instituto de Matematica Relatório Interno No.24, Universidade Estadual de Campinas, Brazil (Jan., 1977).
2. Meskhi, S., Fuzzy propositional logic (an algebraic approach), Bulletin of the Section of Logic, Polish Academy of Sciences 6, 1 (1977), pp.9-14.
3. Moisil, G. C., Notes sur les logiques non-Chrysippiennes, Ann.Sci.Univ. Jassy, 27 (1941) pp.86-98.

A PROPOSED INTERPRETATION OF ŁUKASIEWICZ'S FOUR-VALUED SYSTEM OF MODAL LOGIC

Stanley J. Krolikoski

University of Kentucky
Lexington, Kentucky

In "A system of modal logic"¹ Łukasiewicz introduced his four-valued system \mathcal{L} , a system which was intended to be interpreted modally, but which has, because of certain of its features, resisted such interpretation. In particular, there is in \mathcal{L} no law of necessitation, i.e. no transformation rule of the form

(1) $\vdash a \rightarrow \vdash \Box a$,

where \Box is a necessity functor. There are, in fact, no theses at all of the form $\vdash \Box a$ able to be deduced in \mathcal{L} . Thus, as likely a candidate for thesishood in a modal system as

(2) $\vdash \Box Cpp$

is not a thesis of \mathcal{L} . Lastly, wffs such as

(3) $\vdash \Box CK\Delta p\Delta q\Delta Kpq$

are able to be deduced in \mathcal{L} , wffs which are very unintuitive if Δ is, as Łukasiewicz wishes, interpreted as a possibility functor.

Prior has suggested² that we interpret \Box and Δ as restricted functor variables ranging, respectively, over (a) the negation and falsum and (b) the verum and assertion functors of standard two-valued logic. There is much to this suggestion, but I wish to continue to interpret \Box and Δ as modal functors, albeit non-standard ones. Whereas it is standard to view a necessity (possibility) functor as a restricted universal (existential) quantifier ranging over the set of possible worlds (p-worlds), I propose to view \Box (Δ) as a restricted universal (existential) quantifier ranging over the set which contains all p-worlds and all impossible worlds (i-worlds). Thus, while a proposition is ordinarily said to be necessary iff it is true in all p-worlds, a proposition beginning with \Box will be true iff it is true in all p-worlds and in all i-worlds. Likewise, a proposition beginning with Δ will, on my interpretation, be true iff it is true in at least one member of the set of all worlds, rather than in at least one member of the set of p-worlds, as is the case with standard possibility functors.

The advantage of this interpretation is that in using it we can account for the odd features of \mathcal{L} mentioned above, without reducing the system to non-modal status. Consider, first, that for any logical truth, a , i.e. for any proposition true in all p-worlds, it is easy to specify an i-world in which a is false. For example, Cpp is true in all p-worlds, but is false in the i-world in which (4) If grass is green, then grass is green is false. Thus, we would not expect to be able to assert, in an intuitive logical system, that any proposition was true in all p-worlds and all i-worlds. But, given my interpretation of \Box , that

is exactly what a thesis such as (2) would do.

Hence, if we read \Box as I suggest, we need not be surprised that there are no theses of the form

$\vdash \Box a$ in \mathcal{L} , nor need we be surprised that (1), which would permit the deduction of such theses, is absent from \mathcal{L} .

Further, consider that (3) is unintuitive in a standard modal system because we can imagine situations where p and q are both possible (making $K\Delta p\Delta q$ true) but not compossible (making ΔKpq false). But this objection does not work, given my interpretation of Δ , since even where p and q are both possible but not compossible, i.e. where there conjunction is impossible, ΔKpq is still true, since Kpq holds in at least one of the set of all worlds (viz. in an i-world). Thus, given my interpretation of Δ , one would be surprised if (3) was not a thesis of \mathcal{L} .

Łukasiewicz would surely have disputed my interpretation of \Box and Δ , but I see no other way of accounting for \mathcal{L} 's apparent oddness without denying its modal status.

¹in Journal of Computing Systems Vol. I (1953) pp. 111-149

²"The interpretation of two systems of modal logic" in Journal of Computing Systems Vol. IV (1954) pp. 626-630

SERGE FERRINE

33 Rue Jacquard à METZ FRANCE

Abstract - Some Post algebras are nothing but commutative unitary algebras L over a finite field K where Fermat theorem holds in L .

I. INTRODUCTION

In his article (1) Epstein defines a Post n -algebra as a distributive lattice L with zero 0 and unit u satisfying the following conditions :

Axiom 1 : For every element $x \in L$ there exist n elements $C_0(x), C_1(x), C_2(x) \dots C_{n-1}(x)$

which are pairwise disjoint and whose supremum is u .

Axiom 2 : These exist n fixed elements of L denoted $0=e_0, e_1, \dots, e_{n-1}=u$ with the properties :

(a) The elements form a chain, with $e_{i-1} \leq e_i$ for $1 \leq i \leq n-1$

(b) If $x \in L$ and $xe_1=0$ then $x=0$

(c) If $x \in L$ and for some $i, xe_{i-1}=e_i$ then $x=e_i$

Axiom 3 : For every $x \in L, x = \bigvee_{i=0}^{n-1} e_i C_i(x)$
And he shows

Theorem 16 : The Post algebra L is isomorphic with the set of all continuous n -valued functions on a totally disconnected compact Hausdorff space.

He gives the example of a commutative ring of characteristic p such that for every $x \in R$ we have $x^p = x$. This is the general case.

II. SOME POST ALGEBRAS ARE K -ALGEBRAS.

Let L be a Post n -algebra with a power $n=p^t$ of a prime number p of elements $e_0, e_1 \dots e_{n-1}$. Using the theorem 16 mentioned upper with the n values of the functions taken in the field K with n elements and the discrete topology, L becomes a commutative unitary K -algebra, and the Fermat theorem in K extends in L :

$$\forall x \in L \quad x^n = x$$

III. RECIPROCAL :

Let L be a commutative algebra over the field K . Let us suppose that K have n elements and that L is satisfying :

$$\forall x \in L \quad x^n = x$$

A character of L is a K - morphism from L to

K . The set of all characters is denoted by $X(L)$. The hull kernel topology on $X(L)$ defined in (2) makes a totally disconnected compact Hausdorff space and the K -morphism of Gelfand $G:L \rightarrow F(X(L),K)$ is an isomorphism from L onto the K -algebra of all continuous functions from $X(L)$ to K with the discrete topology. If K is now ordered by : $e_0 < e_1 < e_2 < \dots < e_{n-1}$

L can be ordered by :

$\forall x, y \in L \quad x \leq y \Leftrightarrow \forall z \in X(L) \quad z(x) \leq z(y)$
and becomes now a Post algebra.

IV. APPLICATIONS

The description given upper shows that when

$n=p^t$ is the power of a prime number p , a Post n -algebra L is the same as a commutative unitary algebra over a finite field with n elements K , the algebra satisfying the Fermat theorem :

$$\forall x \in L \quad x^n = x$$

(1) We have a theoretical link with algebraic coding theory (Cf(3)), because Post algebras are the same as algebras used in algebraic coding.

(2) If $n=2$ we find a classical result about Boolean algebras.

(3) If $L = K[X_1, \dots, X_r]$ where for every $i=1, \dots, r, X_i^n = X_i$, we find $X(L) = K^r$ and L is isomorphic with the set of the functions from

K^r to K . This is the reason why the operations defining K form a functionally complete set of operations (Cf(4)).

Conversely it is possible with Van der Monde matrix to give the polynomial expression of any function from K^r to K . This can be done with the polynomial expressions of the elementary Dirac functions which are precisely the $C_0(x), C_1(x), \dots, C_{n-1}(x)$.

V. BIBLIOGRAPHY

- (1) G. EPSTEIN The lattice theory of Post algebras. Trans.Amer.Math.Soc.May 1960,Vol195, N°2,pp.300-317.
- (2) A. GUICHARDET Leçons sur certaines algèbres topologiques. Gordon & Breach.
- (3) J.VAN LINT Coding theory Springer Verlag N° 201 (Ergebnisse der Mathematik)
- (4) IVO G. ROSENBERG Completeness properties of multi valued logic algebras CH 6 in
- (5) DAVID C. RINE Computer Science and multiple valued logic. North Holland.

Ladislav J. Kohout *

University College Hospital Medical School, London, and
 Man-Machine Systems Laboratory, University of Essex, U.K.

Abstract

The paper presents a formulation of a combinatorial multi-valued protection model which is a special case of the possibilistic protection models developed by the author. A class of 5-valued functionally complete logic systems based on the Pinkava multi-valued logic algebras is used as a notational base for the MVL-protection model. The complexity measures given in the paper show that the MVL-protection models are clearly superior to binary ones.

1. Introduction

The concept of protection structures first arose in computer systems motivated by the problem of potentially shared resources in big time-sharing systems or in big computer networks. Graham and Denning's capability-based model¹ represents a succinct formalization of problems which originated in design of protection hardware (eg MULTICS² or MINIC³) or of operating systems.¹² Similar protection problems also arise in data-base design, in particular of those driven by data-interrupts^{4,5}. Examples of data-interrupts are currently found in "artificial intelligence" languages (such as PLANNER, CONNIVER, PLASMA) where protection against unwanted interactions of concurrent processes requires urgent attention⁶. In the robotics and in the artificial intelligence based approach to modelling of perceptual-motor skills^{7,8} similar issues of protection also appear. In this context protection structures⁹ represent necessary constraints on sub-systems, imposed in order to maintain the basic functions of the system as a whole and prevent them from being impaired by the independent and uncoordinated actions of its individual parts.

The global dynamics and stability of general protection models can be analyzed by means of the apparatus of modal logics and of generalized topologies^{10,11}, but the issue is very complex in its most general case. However a special class of these problems represented by non-sequential (combinatorial) protection structures can be more easily analysed by means of finite many-valued logics. The multi-valued logics based approach may be of some practical interest because the combinatorial protection structure can be used to describe a rather important class of protection problems appearing in some practical applications (such as the use of pro-

tection models for the description of parallel processes or the problems invoked in the medical field concerned with the control of muscles and of movement rehabilitation)

The purpose of this paper is to formulate a combinatorial multi-valued logic protection model (MVL-protection model) and to demonstrate its advantages over a binary logic protection model (BL-protection model). Section 2 presents a partial functionally-complete multi-valued logic system, which is used in section 3 to construct a class of functionally complete 5-valued systems. These systems form a notational base for MVL-protection models. Section 4 gives an algebraic formulation of combinatorial protection experiments. This formulation is in its essence an algorithm for the inference (identification) of the structure of combinatorial protection models from experimental protection data. In section 5, the algebraic model of the previous section is embedded in a MVL-protection model, by means of 5-valued logic normal forms. The paper is concluded with a comparison of the complexity of MVL- and BL-protection models and the advantage of a multi-valued formulation is demonstrated.

2. The Pi-Algebras

The Pinkava partial functionally complete multi-valued logic systems^{13,14,15} have been used to construct a number of multi-valued functionally complete logics utilized in several computing and modelling applications^{16,8,11}. Previously, the standard PI-systems¹⁶ had been used in protection modelling⁸. However, with the progress of the work in this area, it has become apparent that the generalized PI-systems^{16,17} (PI-algebras) offer a considerable advantage over the standard ones in the protection applications. The proofs of all theorems on PI-algebras appearing in this section can be found in¹⁷.

The generalization of the PI-systems presented here consists of the representation of the connectives as algebraic groupoids (Kohout¹⁶, theorem 10) and of the cyclic negation by means of the axioms of the cyclic ordering (see the Appendix) familiar from their use in the set-theoretical topology.

The cyclic shift function of the standard systems can be generalised if the discrete cyclic order is used instead. Only finite systems are considered here. In this case the discrete cyclic order can always be defined.

*Requests for offprints should be sent to: Dept. of Electrical Engineering Science, University of Essex, Colchester, Essex, U.K.

Definition 2.1

Let P be a cyclically ordered set and $a \in P$, $b \in P$. Let a precede b and $\exists c$ such that $(a, b, c) \in \mathcal{C}$. We say that $\{a\}$ is the direct predecessor of $\{b\}$ if in the set $\{P \setminus (b)\}$, $\{a\}$ is the last element in $\mathcal{U}_\mathcal{C}(a)$. If every $p \in P$ has a direct predecessor then the cyclic order is called a left-discrete cyclic order. A discrete cyclic order is a cyclic order which is both left and right discrete.

If a is the direct predecessor of b , we write $a \leq b$.

The familiar cyclic negation ($\vec{v} = v+1 \bmod k$) which was used previously, can be generalised:

Definition 2.2 (a discrete cyclic shift function)

Let P be an arbitrary set. If there exists a discrete cyclical order of P , then we define the discrete cyclic shift function corresponding to that cyclical ordering as a mapping ϕ such that:

- 1) $\phi : p \rightarrow p$
- 2) for every $p \in P$ it holds that $p \leq \phi(p)$.

The composition of mappings is defined in the usual way as:

$$\phi^{k+1}(p) = \phi(\phi^k(p)), p \in P$$

Definition 2.3 (distance)

Let $p_1, p_2 \in P$ and ϕ be a discrete cyclic shift function. Then the distance δ of the elements p_1, p_2 with respect to ϕ is the least ordinal such that $\phi^\delta(p_1) = p_2$. We write $\delta_\phi(p_1, p_2)$.

Definition 2.4 (definition of Pi-algebra)

Let Π be an algebra such that

$$\Pi = \langle P, \Diamond, \boxplus, \odot, \phi \rangle \text{ where}$$

- a) P is its carrier
- b) ϕ is a cyclic shift function
- c) $\langle P, \Diamond \rangle$ is an arbitrary groupoid with zero z_\Diamond , without divisors of the zero, and z_\Diamond with the almost-absorbing element a_\Diamond such that

$$a_\Diamond \Diamond p = p \Diamond a_\Diamond = a_\Diamond \text{ for every } p \in P, p \neq z_\Diamond$$

- d) $\langle P, \odot \rangle$ is an arbitrary groupoid with the unit e_\odot

- e) $\langle P, \boxplus \rangle$ is an arbitrary groupoid with a right zero z_{\boxplus} and a right unit e_{\boxplus} .

In order to have a more succinct way of writing let us further introduce the following symbols:

$$\bigodot_{i=1}^n \{x_i\} = x_1 \Diamond x_2 \Diamond x_3 \Diamond \dots \Diamond x_n, n \text{ finite.}$$

Analogously, we introduce the symbols $\bigodot_{i=1}^n$, $\bigboxplus_{i=1}^n$ for repeated operations \odot, \boxplus respectively.

More generally, we shall write e.g. $\bigodot_{x \in S} x$ for repeated operations of taking all elements from an arbitrary set S .

$$\text{Let further } \bigodot \{ \phi^K(v) \} = v \Diamond \phi(v) \Diamond \phi^2(v) \Diamond \dots$$

$$\bigodot \phi^K(v).$$

$\bigodot \{ \phi^K(v) \} - \phi^V(v)$ is the abbreviation for

$$v \Diamond \phi(v) \Diamond \dots \Diamond \phi^{V-1}(v) \Diamond \phi^V(v) \Diamond \dots$$

Definition 2.5

$$\psi_K(v) = \begin{cases} a_\Diamond & \text{iff } v=K \\ z_\Diamond & \text{iff } v \neq K \end{cases}$$

Lemma 2.6

The function that has the constant value c , where c is any element of P , is given by:

$$c(v) = \phi^K \left[\bigodot \{ \phi^X(v) \} \right] \text{ where } K = \delta(z_\Diamond, c)$$

Lemma 2.7

The characteristic function is generated by the expression $\psi_c(v) = \bigodot \{ \phi^X(v) \} - \phi^{*K}(v)$, where $K = \delta(z_\Diamond, c)$ and ϕ^* is an inverse cyclic shift function obtained by the substitution of ϕ^* for ϕ in Definition 2.2.

Lemma 2.8

If $\delta(a_\Diamond, e_{\boxplus}) = \delta(z_\Diamond, z_{\boxplus})$ then any function $f(v_1, v_2, \dots, v_n)$ of a many-valued logic system may be expressed by means of a formula of the following type:

$$f(v_1, v_2, \dots, v_n) = \left(\bigodot_{\ell=1}^n \phi^{\delta_2}(\phi^{\delta_1^*}(c_Y) \boxplus \phi^{\delta_1} \left[\bigodot_{\ell=1}^n \psi_{a_\ell}(v_\ell) \right] \right) \{ \forall f(a_1, a_2, \dots, a_n) | f \neq e_\odot \}$$

where $\delta_1 \stackrel{\text{def}}{=} (a_{\diamond}, e_{r_{\boxplus}})$, $\delta_2 \stackrel{\text{def}}{=} (z_{\boxplus}, e_{\odot})$.

Remark: In the above formula $(f \odot \dots)$ means the repeated operation for such substitutions $a_1, a_2, \dots, a_n, a_j \in P$, of the variables v_1, v_2, \dots, v_n , for which $f \neq e_{\odot}$.

The distance δ_2^* is the inverse of the distance δ_2 .

Theorem 2.9

Any Pi-algebra is functionally complete if the following condition is satisfied:

$$(z_{\diamond}, z_{\boxplus}) = \delta_1, \text{ where } \delta_1 \stackrel{\text{def}}{=} (a_{\diamond}, e_{\boxplus}).$$

Theorem 2.10

If the right zero $z_{r_{\boxplus}}$ is also the zero and the right unit $e_{r_{\boxplus}}$ is also the unit of the groupoid $\langle P, \boxplus \rangle$ then the following holds:

$$\odot_{\phi}^{\delta_2} \{ \odot_{\phi}^{\delta_2^*} (C_Y) \boxplus \phi^{\delta_1} [\bigoplus_{l=1}^n \psi_{\alpha_l} (v_l)] \} =$$

($i \neq e_{\odot}$)

$$= \odot_{\phi}^{\delta_2} \{ \odot_{\phi}^{\delta_2^*} (C_Y) \boxplus [\bigoplus_{l=1}^n \phi^{\delta_1} (\psi_{\alpha_l} (v_l))] \}$$

($f \neq e_{\odot}$)

3. Design of Functionally Complete Systems Suitable for the Description of Protection Structures

3.0 Brief description of the method

Design of some many-valued systems suitable for the description and verification of finite protection structures will be outlined in this section. This will be based on generalised Pinkava many-valued systems that were described in detail in the previous section. Although it is possible to base this design on standard Pinkava systems (cf. Kohout & Gaines 1976)¹¹, the use of generalised systems offers some advantages as has been demonstrated in Kohout (1976)⁹.

Besides the logical properties of protection structures, there are always extra-logical conditions which it is desired that a system should meet. If these can all be embodied in one of the connectives, that is called a principal connective.

The design of new systems should consist of the following steps:

- 1) choice of a suitable k determining the number of logical values in a k -valued functionally complete calculus;
- 2) decision concerning the meaning of each logical value in the context of protection structures;
- 3) examining relations between individual values, which are implied by some extra-logical properties of protection structures; these will

determine the additional properties of the principal connective \odot .

- 4) specification, which connectives of the set $\{\diamond, \boxplus, \odot\}$ can be chosen as principal; note that there may exist several choices - in the extreme case any of $\diamond, \boxplus, \odot$ may be chosen as the principal connective;
- 5) choice of the best principal connective;
- 6) the full specification of the other connectives that with the chosen principal connective form a functionally complete system; these are usually chosen in such a way that they possess some good algebraic properties advantageous for minimization.

In the following sections, the above mentioned method will be applied to the design of a 5-valued system for verification of protection structures.

3.1 Choice of a suitable number of logical values and of a principal connective

In the description of a protection system it is required to take into account at least two types of conditions; that is - what properties must be present and which properties are not permitted. However, in practice, no structure will be fully defined, hence don't care conditions will be introduced. For this reason the many-valued logics used should be at least ternary logics as the three following types of conditions have to be taken into account:

- a) conditions describing capabilities, passes and permits which must be present and which are necessary for a proper functioning of the whole system;
- b) conditions specifying which capabilities, passes and permits would violate the protection status of the whole system and therefore must not be present;
- c) don't care conditions specifying which capabilities, passes and permits can be introduced by individual participants.

This leads to a logic with the following interpretation of values:

$$\{\text{necessary, don't care, impossible}\} = \{n, d, i\}.$$

Structural models based on a three-valued logical calculus with the values given the interpretation, described above can be used for verification of protection systems, if suitable reduction techniques of Gentzen-style are employed.

Another possibility, which will be employed in this work is based on direct semantic techniques employing normal forms. For this, it is necessary to amend the logical calculus by introducing a new logical value v ; which specifies when the violation of some logical conditions characterising a protection structure occurs. It follows from the above description of the value set that the logical conditions describing a protection structure will be mapped into $\{d, i, n\}$. Hence, in order to be able to detect (in the eventual normal form), the violation of the conditions, when any two incompatible condi-

tions are combined by the principal connective, the result should be mapped into v.

"Don't care" as a logical value can designate the cases where some relation or a condition appearing in the protection model is irrelevant with respect to the function of protection. In practice, there will also appear vagueness due to lack of exact information concerning a particular protection structure, the situation when we only know that some relationship between participants is possible. The algebraic properties binding the latter logical value in the principal connective are stronger than those binding the d logical value but weaker than the conditions binding the other logical values. In order to take this situation of mere possibility into account, a new 5-valued principal connective is introduced:

\odot	d	p	n	i	v
d	d	p	n	i	v
p	p	p	n	i	v
n	n	n	n	v	v
i	i	i	v	i	v
v	v	v	v	v	v

(3.1-2)

3.2 The selection of a functionally complete system.

Next step in the design of a complete system is to assign the logical values of the principal connective \odot to a groupoid from the set of connectives $\{\diamond, \boxplus, \odot\}$. There are possible the following assignments of values for both the 5-valued principal connectives from the previous section, and the 4-valued system from⁹;

- a) $d=e_{\odot}$ (3.1-2)
 b) $d=e_{\boxplus}$; $v=z_{\boxplus}$.

It can be seen from this assignment that two distinct classes using all connectives can be obtained;

- a) $\{\odot, \diamond, \boxplus, \rightarrow\}$ where $\odot \equiv \odot$ (3.2-2)
 b) $\{\diamond, \odot, \odot, \rightarrow\}$ where $\odot \equiv \boxplus$

It is obvious that it is possible to choose as the non-principal connectives of each system arbitrary connectives of each corresponding type. However, it has been pointed out that it may be advantageous to select complete systems which have some additional special properties. For example we can choose a pair of connectives which define a lattice. If the logical value v is taken as the greatest element of the lattice and the logical value d as the smallest element, the dual connective \odot_d to \odot will be defined by the following table:

\odot_d	d	p	n	i	v
d	d	d	d	d	d
p	d	p	p	p	p
n	d	p	n	p	n
i	d	p	p	i	i
v	d	p	n	i	v

(3.2-3)

The "zeroes", "units" and "almost-absorbing element" can be assigned in the following way:

- a) $z_{\odot_d}=v$, $z_{\odot_d}=d$
 b) $e_{\odot_d}=d$, $e_{\odot_d}=v$, $a_{\odot_d}=p$ (3.2-4)

Note that for the 4-valued system presented in⁹ no almost-absorbing element

For the pair $\{\odot, \odot_d\}$ there is a greater degree of freedom in their assignment to the set $\{\diamond, \boxplus, \odot\}$. Now, several examples of complete system will be given for the illustration of the method.

Example (3.2-5):

$$\odot \equiv \boxplus, \odot_d \equiv \odot (\equiv \diamond)$$

$$v=z_{\boxplus}=e_{\odot}, d=e_{\boxplus}$$

The normal form is given by the following expression:

$$f(v_1, v_2, \dots, v_n) = \odot_{(f \neq e_{\odot} \equiv v)} \left\{ \odot_{\boxplus} \left[\bigoplus_{l=1}^n \alpha_l (v_l) \right] \right\}$$

The cyclic shift function is given by the following table:

x	d	v	π_1	π_2	p
\bar{x}	v	π_1	π_2	p	d

where the permutations of the logical values i, n are denoted by

$$\pi(i, n) = \pi_1, \pi_2 = \begin{Bmatrix} i, n \\ n, i \end{Bmatrix}$$

Example 3.2-6

$$(P) \equiv \boxplus (\equiv \odot), (P_d) \equiv \boxplus; d = e_{\boxplus} = z_{\boxplus} = e_{\odot}, p = a_{\boxplus}, v = z_{\boxplus}.$$

$$f(v_1, v_2, \dots, v_n) = \boxplus_{f \neq d} \{ \boxplus_{\gamma} \boxplus_{\alpha_l} [\boxplus_{\alpha_l} (v_l)] \} \rightarrow^4 =$$

$$= \boxplus_{f \neq d} \{ \boxplus_{\gamma} \boxplus_{\alpha_l} [\boxplus_{\alpha_l} (v_l)] \} \rightarrow^4$$

where \rightarrow is the cyclic shift function defined in (3.2-5).

The choice of a many-valued system is not influenced only by the requirement for good algebraic properties. Perhaps more important is the requirement that such forms should be chosen from all available normal forms, which give the least complex expressions for a given class of protection structures. One possible simplification is to map into e_{\odot} that logical value which appears the most frequently in a many-valued logical function which is being expressed by the normal form. For example, if the don't care condition is the most frequent, the obvious choice is $d = e_{\odot}$. This value is then omitted when the full canonical form is written down.

The example (3.2-7) that follows clearly demonstrates that the choice of an appropriate normal form can reduce significantly the length of the initial formula to be minimized which consequently leads to the decrease of the computing time needed for the minimization.

Example 3.2-7

Let us compare the canonical normal forms of a 5-valued function $f_1(x, y)$ given in Fig. 1 which are expressed by means of the connectives (P) and (P_d) given the following assignments:

- the assignment of Example 3.2-5;
- the assignment of Example 3.2-6.

Let the cyclic shift function be defined by the substitution ($\pi_1 = i, \pi_2 = n$) into the table of example 3.2-5, in both cases.

Then for (a) we obtain the following normal form:

$$f_1(x, y) = p \boxplus [\vec{\psi}_0(x) \boxplus \vec{\psi}_0(y) \boxplus \vec{\psi}_4(y) \boxplus \vec{\psi}_4(x) \boxplus \vec{\psi}_0(y)$$

$$\boxplus \vec{\psi}_4(y) \boxplus \vec{\psi}_0(x) \boxplus \vec{\psi}_0(y) \boxplus \vec{\psi}_2(y) \boxplus$$

$$\boxplus \vec{\psi}_4(y) \boxplus \vec{\psi}_2(y) \boxplus \vec{\psi}_2(x) \boxplus \vec{\psi}_4(y) \boxplus \vec{\psi}_2(y)] \odot$$

$$\odot n \boxplus [\vec{\psi}_0(x) \boxplus \vec{\psi}_3(y) \boxplus \vec{\psi}_4(x) \boxplus \vec{\psi}_3(y)].$$

For (b) the form is given by

$$f_1(x, y) = \vec{p} \boxplus [\vec{\psi}_0(x) \boxplus \vec{\psi}_0(y) \boxplus \vec{\psi}_4(y) \boxplus \vec{\psi}_4(x) \boxplus \vec{\psi}_0(y)$$

$$\boxplus \vec{\psi}_4(y) \boxplus \vec{\psi}_0(x) \boxplus \vec{\psi}_2(y) \boxplus$$

$$\vec{\psi}_4(y) \boxplus \vec{\psi}_2(y) \boxplus \vec{\psi}_2(x) \boxplus \vec{\psi}_0(y) \boxplus \vec{\psi}_4(y)$$

$$\boxplus \vec{\psi}_2(y) \boxplus \vec{\psi}_0(x) \boxplus \vec{\psi}_3(y) \boxplus \vec{\psi}_4(x)$$

$$\boxplus \vec{\psi}_3(y) \boxplus \vec{\psi}_0(x) \boxplus \vec{\psi}_1(y) \boxplus$$

$$\vec{\psi}_4(x) \boxplus \vec{\psi}_1(y) \boxplus \vec{\psi}_3(x) \boxplus \vec{\psi}_0(y) \boxplus \vec{\psi}_1(y) \boxplus$$

$$\vec{\psi}_4(y) \boxplus \vec{\psi}_3(y) \boxplus \vec{\psi}_2(y) \boxplus \vec{\psi}_1(y) \boxplus \vec{\psi}_2(x)$$

$$\boxplus \vec{\psi}_3(y) \boxplus \vec{\psi}_1(y) \boxplus \vec{\psi}_1(x) \boxplus \vec{\psi}_0(y) \boxplus \vec{\psi}_4(y)$$

$$\boxplus \vec{\psi}_3(y) \boxplus \vec{\psi}_2(y) \boxplus \vec{\psi}_1(y) \boxplus \vec{\psi}_4(y)].$$

It is obvious that (a) represents a considerable saving, for in this case the number of elementary expressions between the connectives is equal to 21 whilst in (b) is equal to 42.

4. Combinatorial Protection Models

A set of processes mutually cooperating or competing perform their activities according to some protection rules specifying what is permitted and what is forbidden. General formulation by means of sequential capability-based models is unnecessary in the case we are interested only in the question whether some concurrent processes violate given protection rules. This situation may appear in certain computing protection models¹² as well as in some applications of the protection structures to modeling of biological and medical systems³. This class of problem can be described by combinatorial protection models. These models can be either formulated directly or built up from combinatorial protection experiments. The latter approach is preferable, for it allows us to infer (or identify) the structure of a protection model from the experimental data. These data are described by means of protection experiments which represent families of samples collected repeatedly on the systems, the protection status of which is being analyzed.

Given a set of processes $X = \{x_i\}$ we are interested in a description of a family of subsets of X which are needed in order to perform certain activity. This activity is characterized by a task descriptor T .

Definition 4.1 (task descriptor)

A task descriptor \bar{T} , associated with an activity τ performing a set of tasks T , is a partition of T into two disjoint subsets

$$\bar{T} = \langle T_p, T_f \rangle$$

where T_p is the subset of permitted tasks and T_f the

subset of forbidden tasks, of \bar{T} .

Definition 4.2 (completion of an aim)

An activity τ completes its aim if all associated T_p are achieved. It completes it correctly if none of the associated tasks T_f have been performed during this act of completion.

In the process of performing a set of tasks T , some processes are active and some inactive. Each activity is completed by the activation or disactivation of some processes from X . Generally, there is no unique way of completion of an aim. The same completion can be achieved by many distinct patterns of activating or disactivating of processes. In those special instances, when the order of activation of processes is irrelevant with respect to a correct completion of an aim, the activity of completion can be described by means of combinatorial protection experiments. These can be conveniently analyzed by means of finite many-valued logics.

Definition 4.3 (protection experiment).

A protection experiment is a quadruple

$$PE = \langle \bar{T}, X, s, \{M_i\} \rangle$$

where \bar{T} is a task descriptor and $\{M_i\}$ is an indexed family of protection samples, and $X = \{x_j\}, (j=1, 2, 3, \dots, r)$ is a set of processes; r and s are the numbers of processes and samples, respectively; \emptyset is a set describing the states which the processes can attain (e.g. ACTIVE, INACTIVE, IDLE, etc.).

An i -th protection sample M_i is a tuple

$$M_i = (\Lambda_i, \Omega_i)$$

where $\Lambda_i = \lambda_i(T_p)$ and $\Omega_i = \omega_i(T_f)$ are the ranges of the relations

$$\lambda_i: T_p \rightarrow \times_r \Gamma$$

$$\omega_i: T_f \rightarrow \times_r \Gamma$$

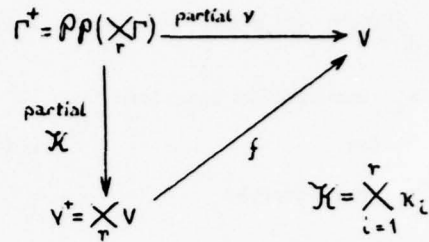
where Γ is defined by $\Gamma = (X \times \emptyset) \cup \{-\}$; the symbol \times denotes a cartesian product. The symbol $\{-\}$ represents a blank symbol. In practice this means that if in a measurement we are not able to identify the pair (x_j, α_j) correctly, we substitute $\{-\}$ instead.

Definition 4.4 (MVL-protection model)

An MVL-protection model of a PE is a mapping

$$f: \times_r V \rightarrow V$$

such that the following diagram commutes:



where:

- V is the set of logical values (e.g. $\{d, p, n, i, v\}$)
- the partial mapping is formed according to the following valuation rules:

$$\Lambda = \bigcap_{i=1}^s \Lambda_i \quad \tilde{\Lambda} = \bigcup_{i=1}^s \Lambda_i \quad A \in \rho\rho\Gamma^r$$

$$Q = \bigcap_{i=1}^s \Omega_i \quad \tilde{Q} = \bigcup_{i=1}^s \Omega_i$$

	Condition	Valuation
b.1.1.	$A \cap Q \neq \emptyset$ and $A \cap \tilde{\Lambda} \neq \emptyset$ implies (don't care, irrelevant)	$v(A) = d$
b.1.2	$A \cap \tilde{Q} \neq \emptyset$ and $(A \cap \tilde{\Lambda}) \equiv (A \cap \tilde{\Lambda}) \neq \emptyset$ (don't care, irrelevant)	$v(A) = d$
b.2	$A \cap \tilde{Q} \neq \emptyset$ violation	$v(A) = v$
b.3	$A \cap \tilde{\Lambda} \neq \emptyset$ and $A \cap \tilde{Q} = \emptyset$ necessary	$v(A) = n$
b.4	$A \cap Q = \emptyset$ and $A \cap \tilde{\Lambda} \neq \emptyset$ possible	$v(A) = p$
b.5	$A \cap (\tilde{Q} \cup \tilde{\Lambda}) = \emptyset$ impossible	$v(A) = i$

- the partial mapping X is a cartesian product of the component mappings κ_i ,

$$X = \times_{i=1}^r \kappa_i$$

A component mapping κ_i must obey the following conditions:

	Condition	Valuation
c.1	$\bigcap_{j=1}^s \text{Proj}_{\alpha_j}(\Lambda_i) \neq \emptyset$ x_i necessarily active	$\kappa_i(\alpha_j) = n$
c.2	$\bigcup_{j=1}^s \text{Proj}_{\alpha_j}(\Lambda_i) \neq \emptyset$ x_i possibly active	$\kappa_i(\alpha_j) = p$

- c.3 $\bigcap_{i=1}^s \text{Proj}_{\bar{a}_j}(\Lambda_i) \neq \emptyset$ $\kappa_i(\bar{a}_j)=i$
 x_i necessarily inactive
- c.4 $\alpha_j=\{-\}$ $\kappa_i(\alpha_j)=d$
 x_i not observed

The set $\Theta=\{\alpha_j^0, \bar{\alpha}_j\}$ and $\alpha_j^0(x_j, \alpha_j^0)$, $\bar{\alpha}_j(x_j, \bar{\alpha}_j)$. The states of processes α_j have assigned the meanings "ACTIVE" or "INACTIVE".

Remark 4.5 (on binary logic models)

A BL-model can be obtained by substituting $V=\{0,1\}$ and defining V^+ as $V^+=\bigcup_{p=2rs} V$, where $p=2rs \cdot \text{card}(\Theta)$. A reduced binary model (RBL-) can be obtained by splitting the mapping into two parts, giving each part slightly different valuation rules and mapping into V^+ . An example of protection models is given in Appendix 2 and the complexity of both an MVL- and RBL- protection model is compared.

5. Comparison of the Complexity of MVL- and BL- Models.

A canonic table of a logic function f is defined as a tuple (V^+, V) - for an example see Fig.1.

Complexity of the canonical normal form will be a function of the complexity coefficient C of the corresponding canonic table (i.e. of the 'length' of the table).

We obtain the following formulas for $\text{card}(\Theta)=2$:

$$\begin{aligned} C(\text{MVL}) &= 5^r - e, \text{ where } e \text{ is } \text{card}(A) \text{ such that} \\ & f(A) = e_{\ominus}, A \subseteq V^+; \\ C(\text{BL}) &= 2^{4rs} - e \\ & (rq + (1 + (\sqrt{\log_2 s}))) \\ C(\text{RBL}) &= 2 \end{aligned}$$

It can be seen that the complexity of MVL-models does not depend on the numbers of protection samples M_i , whereas it depends on s for the binary models. This can be explained by the fact that some, for the purpose of the protection analysis inessential structural information is omitted, when we are coding into an MVL.

6. Conclusion

An MVL-protection model aiding the analysis of combinatorial protection structures has been presented in this paper. It has been shown that the complexity of this model is considerably smaller than of corresponding binary models.

Recently, some reports on application of fuzzy sets of protection problems have appeared. It should be noted that the MVL-model presented in this

paper can be easily 'fuzzified' by means of the method described in 18.

Appendix 1

The definition of a cyclical ordering and two auxiliary Lemata needed in section 2 of the paper are given here. More details and the proofs can be found in E.Čech (1969), Point Sets, published by Academia, Prague (distributed by Academic Press, New York).

Definition (Čech, 1969)

A cyclical ordering of a set P is a subset \mathcal{C} of the set $P \times P \times P$ satisfying the following conditions:

- 1) $(a, b, c) \in \mathcal{C} \Rightarrow (b, c, a) \in \mathcal{C}$
- 2) $(a, b, c) \in \mathcal{C}$ and $(b, a, c) \in \mathcal{C}$ never hold simultaneously
- 3) if neither $(a, b, c) \in \mathcal{C}$ nor $(b, a, c) \in \mathcal{C}$, then two of the elements a, b, c are equal.
- 4) $(a, b, c) \in \mathcal{C}, (a, c, d) \in \mathcal{C} \Rightarrow (a, b, d) \in \mathcal{C}$.

Lemma A

Let P be a cyclically ordered set and let $a \in P$. If $x \in P \setminus \{a\}, y \in P \setminus \{a\}$, we say that x precedes y if and only if $(a, x, y) \in \mathcal{C}$. Then we shall write $x \prec y$. This defines an ordering of the set $P \setminus \{a\}$, which will be denoted $\mathcal{U}_a(a)$. For the proof that this is an ordering see Čech (1969), p. 34.

Lemma B

Let $a \in P$ and let there be given an ordering of the set $P \setminus \{a\}$. Then there is exactly one cyclical ordering \mathcal{C} of the set P such that the given ordering of the set $P \setminus \{a\}$ coincides with $\mathcal{U}_a(a)$. Proof: see Čech (1969), p.35.

Definition

Let \mathcal{C} be a cyclical ordering of a set P . Define $P \times P \times P$ as follows:

$$(a, b, c) \in \mathcal{C}^* \Leftrightarrow (c, b, a) \in \mathcal{C}$$

Then \mathcal{C}^* is the inverse cyclical ordering to \mathcal{C} .

Appendix 2 - an example: inference from experimental data and the construction of an MVL-model and its comparison with BL- and RBL-models

This example describes a protection experiment $PE = (\bar{T}, \{x_1, x_2\}, 4, \{M_i\})$ on the set of processes $X = \{x_1, x_2\}$. $X \times \Theta = \{(x_1, \alpha^0), (x_1, \bar{\alpha}^0), (x_2, \alpha^0), (x_2, \bar{\alpha}^0)\}$, where (x_1, α^0) means the process x_1 is activated, $(x_1, \bar{\alpha}^0)$ means x_1 is not active. Let us denote for simplicity $x = (x_1, \alpha^0)$ $\bar{x} = (x_1, \bar{\alpha}^0)$ $y = (x_2, \alpha^0)$ $\bar{y} = (x_2, \bar{\alpha}^0)$.

Then the protection samples M_i are given in the following table:

i	Λ_i	Ω_i
1	$(-,y),(x,-),(x,y),(-,-);$	$(x,\bar{y}),(\bar{x},y)$
2	$(-,y),(x,-),(x,y)$	$(x,\bar{y}),(\bar{x},y)$
3	$(-,y),(x,-),(x,y)$	$(x,\bar{y}),(\bar{x},y)$
4	$(-,y),(x,-),(x,y)$	$(x,\bar{y}),(\bar{x},y)$

Note, that Λ is the collection of samples of activities of processes leading to desirable tasks and Ω is the collection leading to undesirable activities against which we want to implement a protection mechanism. The MVL-model constructed from this PE is in Fig.1 as the function $f_2(x,y)$.

The complexity measures of the normal forms are:

$$C(MVL)=14 \text{ (for } e_{\odot}=v \text{ we have } e=11)$$

$$C(BL) = 2^{32-e_1}$$

$$C(RBL)=128-e_2$$

After the minimization the models are:

$$MVL(f_2(x,y))=(x \oplus y) \oplus (x \oplus p) \quad (\oplus = \boxplus)$$

$$RBL(f_2(x,y,s_1,s_2,s_3))=s_1[y(x\bar{x}\bar{y}\bar{x}\bar{x})Vx(\bar{y}y\bar{y}\bar{y})V$$

$$Vxy\bar{x}\bar{y}\bar{y}(xy\bar{x}\bar{y}\bar{x}\bar{y}\bar{x}\bar{y})\bar{s}_2\bar{s}_3].s_1[\bar{x}y\bar{x}\bar{y}\bar{y}Vxy\bar{x}\bar{y}]^-.$$

Here the triple (s_1,s_2,s_3) is binary coded index i in \mathcal{M}_i . Note that this represents some redundant information from the point of view of the protection analysis but omitting this triple altogether would unfortunately lead to the destruction of the information that is essential for the analysis.

Remark: In the RBL example, $(x_1,\bar{\bar{y}})=x$ and $(x_1,\bar{\bar{y}})=x$ etc. The bar above a letter is used to denote the binary negation.

x	y	$f_1(x,y)$	$f_2(x,y)$
1	d d	p	p
2	d p	p	p
3	d n	v	n
4	d i	i	i
5	d v	<input checked="" type="checkbox"/>	v
6	p d	p	p
7	p p	p	p
8	p n	n	n
9	p i	i	i
10	p v	v	v
11	n d	<input checked="" type="checkbox"/>	n
12	n p	<input checked="" type="checkbox"/>	n
13	n n	<input checked="" type="checkbox"/>	n
14	n i	<input checked="" type="checkbox"/>	v
15	n v	<input checked="" type="checkbox"/>	v
16	i d	i	i
17	i p	i	i
18	i n	<input checked="" type="checkbox"/>	v
19	i i	i	i
20	i v	<input checked="" type="checkbox"/>	v
21	v d	<input checked="" type="checkbox"/>	v
22	v p	<input checked="" type="checkbox"/>	v
23	v n	<input checked="" type="checkbox"/>	v
24	v i	<input checked="" type="checkbox"/>	v
25	v v	<input checked="" type="checkbox"/>	v

Fig. 1

References

1. Graham, G.S. & Denning, P.J. (1972) Protection principles and practice, Proc. Spring Joint Comp. Conf. 40, AFIPS Press, New Jersey, 417-429.
2. Organick, E.I. (1972) The Multics System, MIT Press.
3. Gaines, B.R., Facey, P.V., Williamson, F.K., & Maine, J.A. (1974) Design objectives for a deascriptor-organised minicomputer, Eurocomp 74, Online Ltd. London, 29-45.

4. Morgan, H.L. (1970) An interrupt-based organization for management information systems CACM 13, 734-739.
5. Gaines, B.R., Facey, P.V., & Sams, J. (1974) An interactive display-based system for gilt-edged security broking, EUROCOMP 74 Online Ltd., London, 155-169.
6. Hewitt, C. (1969) PLANNER: a language for manipulating models and proving theorems in a robot, Proc. of Internat. Joint Conf. on Artificial Intelligence, Washington.
- Hewitt, C. (1971) Procedural imbedding of knowledge in PLANNER, Int. Conf. on Artificial Intelligence, London.
- Hewitt, C. (1974) Protection and synchronisation in actor systems, Working Paper 83, Artificial Intelligence Laboratory, MIT, Cambridge, Mass.
- Hewitt, C. (1975) Protection and synchronisation in actor systems, Working Paper 90, Artificial Intelligence Laboratory, MIT, Cambridge, Mass.
7. Kohout, L. (1976) The hierarchic control of movement, an invited contribution. AISB meeting on Human & Robot Behaviour, Leicester Polytechnic, April 1976.
8. Kohout, L.J. (1976a) Representation of functional hierarchies of movement in the brain, Int. J. Man-Machine Studies, 8, 699-709.
9. Kohout, L.J. (1976) Application of multi-valued logics to the study of human movement control and of movement disorders, Proc. 6th Int. Symp. Multiple-Valued Logic, IEEE 76CH1111-4C, 224-231.
10. Kohout, L.J. & Gaines, B.R. (1975) The logic of protection, Lecture Notes in Computer Science, 34, 736-751, Springer, Berlin-New York.
11. Kohout, L.J. & Gaines, B.R. (1976) Protection as a general systems problem, Int. J. Gen. Syst. 3, 3-23.
12. Harrison, M.A. (1975) On models of protection in operation systems, in Bečvář, J. (ed), Mathematics in Computer Science, Lecture Notes in Comp. Sci., 32. Springer, Berlin, 46-60.
- Harrison, M.A., Ruzzo, W.L., & Ullman, J.D., (1976) Protection in operating systems, Comm. Assn. Comp. Mach. 19 461-471.
13. Rosenberg, I.G. (1976) Some algebraic and combinatorial aspects of multiple-valued circuits, In Proc. 6th Symp. Multiple-Valued Logics.
14. Pinkava, V. (1975) Some further properties of the Pi-logics, Proc. 1975 Int. Symp. Multiple-Valued Logic, IEEE 75CH0959-7C, 20-26.
- Pinkava, V. (1976a) Minimization of Pi-logics, in Mamdani, E.H. & Gaines, B.R. (eds), Discrete Systems and Fuzzy Reasoning, Queen Mary College, London, (workshop proceedings).
15. Pinkava, V. (1978) On functionally complete Pi-algebras, Studia Logica, in print, No. 2.
16. Kohout, L.J. (1974) The Pinkava many-valued complete logic systems and their applications in the design of many-valued switching circuits, in Proc. 1974 Int. Symp. Multiple-Valued Logic, IEEE 74CH0845-8C, 261-284.
17. Kohout, L.J. & Pinkava, V. (1976) Functional completeness of Pi-algebras and its relevance to biol. modelling, in Mamdani, E.H. & Gaines, B.R. (eds), Discr. Syst. & Fuzzy Reasoning, Queen Mary College, London, (workshop proc.)
18. Pinkava, V. (1976) Fuzzification of binary and finite multivalued logical calculi, Int. J. Man-Machine Studies, 8, 717-730.

THE B-TERNARY LOGIC AND ITS APPLICATIONS
TO THE DETECTION OF HAZARDS IN COMBINATIONAL SWITCHING CIRCUITS

Masao Mukaidono

Faculty of Engineering
Meiji University
Ikuta Kawasaki-shi Japan 214

Abstract

Both the steady states and some transient states of switching circuits can be described by B-ternary logic in which the truth values 0, 1 and $1/2$ are used respectively to represent false, true and uncertainty. This paper showed the methods of detecting and identifying various kinds of static hazards contained in combinational switching circuits by means of the canonical forms of the B-ternary logic functions realized by the circuits. Particularly, a method was derived which could algebraically detect all logic hazards contained in the circuits. It was also pointed out that there were some dynamic hazards which were detectable by B-ternary logic.

1. Introduction

D.A.Huffman¹ and E.J.McCluskey², firstly, pointed out hazards contained in switching circuits and discussed procedures for detecting and removing hazards. Since then, many researchers³⁻¹¹ have investigated these subjects. On the other hand, M. Goto¹² proposed that a suitable ternary logic (i.e., B-ternary logic) in which the truth values 0, 1 and $1/2$ are used respectively to represent false, true and uncertainty was applicable to describing uncertain behaviors of relays. It was shown by M. Yoeli⁴ that the B-ternary logic could be utilized to detect static hazards in combinational switching circuits. Furthermore, the method was extended by Eichelberger⁵ for application to hazards with multiple-input changes and hazards in sequential circuits.

Based on some new properties of B-ternary logic clarified recently by M.Mukaidono¹³, the present paper is concerned with the development of new applications of B-ternary logic to detecting and identifying various kinds of hazards caused by multiple-input changes. After formalizing various kinds of static hazards by means of the B-ternary logic in chapter 4, we discuss in chapter 5 procedures for detecting and identifying hazards by utilizing the disjunctive and conjunctive forms of the B-ternary logic functions realized by the circuits. In chapter 6 we show a method for detecting algebraically all logic hazards contained in the circuits by obtaining the corresponding B-ternary

canonical forms. Up to present, it has been considered that the B-ternary logic is powerless in regard to dynamic hazards, but chapter 7 shows that some dynamic hazards are detectable by the B-ternary logic.

The fuzzy logic^{14,15} is an extension of the B-ternary logic and satisfies the same algebraic system. Thus, the fuzzy logic and the B-ternary logic have the same ability to represent the transient behaviors corresponding to hazards of switching circuits¹⁶. This paper uses mainly the canonical forms of B-ternary logic functions, but it needs to be noted that we will obtain the same results even if we use the canonical forms of fuzzy logic functions.

2. Hazards in Combinational Switching Circuits

Usually, we use the binary logic (i.e., Boolean algebra) to design switching circuit networks. Although the binary logic can describe all the steady state behaviors of switching circuits, but it cannot adequately describe transient state behaviors. This is because time factors are ignored in the theory of binary logic, whereas time delay necessarily exists in the real switching circuit between signal input and output. Hazards are one example of such unexpected behavior occurring in switching circuits in a transient state. We shall begin by defining various kinds of hazards in combinational switching circuits according to M. Yoeli⁴ and E.B.Eichelberger⁵.

[Definition 1]⁵ A combinational switching circuit F contains a static hazard due to an input change (in this paper input change involves the changing of one or more input variables) iff

- (1) the output of F before the change is equal to the output after the change,
- (2) during the change spurious pulses may appear in the output.

Concerning the above definition, it should be noted that the statement "A circuit contains a static hazard" means that it is possible for the circuit to contain a spurious pulse due to an input change and does not mean that a spurious pulse is always generated in the output. That is, Definition 1 is based on the most pessimistic cases. This means that it is always possible to produce a spurious pulse by inserting delay elements in certain places of the circuit.

Let F denote a n -input variables combinational switching circuit and f denote the binary logic function (i.e., switching function) realized by F . We designate an input before change and an input after change by $a=(a_1, \dots, a_n)$ and $b=(b_1, \dots, b_n)$ respectively, where a and b are elements of V_2^n and $V_2=\{0,1\}$. Using this notation, the condition (1) of Definition 1 is written as $f(a)=f(b)$.

[Definition 2]⁴ A static hazard in F generated by an input change from a steady state a to another steady state b (henceforth, written as $a \rightarrow b$) is said to be a static 0 (1) hazard iff $f(a)=f(b)=0$ ($f(a)=f(b)=1$).

[Definition 3]⁵ A static hazard in F generated by an input change $a \rightarrow b$ is said to be a logic hazard iff all values of f are equal to each other for the inputs a, b and all the inputs which may result during the change $a \rightarrow b$.

[Definition 4]⁵ A static hazard in F generated by an input change $a \rightarrow b$ is said to be a function hazard iff there is an input state a' such as $f(a) \neq f(a')$ in the inputs which may result during the change $a \rightarrow b$.

Function hazards are inherent in the binary function f and do not depend on the construction of the circuit F . It was shown by E.B.Eichelberger that function hazards could not be eliminated but logic hazards could be removed by using all prime implicants in disjunctive form (sum of product form).

[Definition 5] A combinational switching circuit F contains a dynamic hazard due to an input change $a \rightarrow b$ iff

- (1) $f(a) \neq f(b)$,
- (2) during the change spurious pulses may appear in the output.

3. B-ternary Logic¹³

Both the steady states and some transient states of switching circuits can be described by B-ternary logic in which the truth values 0, 1 and 1/2 are used respectively to represent false, true and transient state, that is, uncertain as to 0 or 1. In the present chapter we describe only pertinent parts of the theory of B-ternary logic. If readers wish to know more about details of the theory of B-ternary logic, refer to (13).

The B-ternary logic is a ternary logic system on which logical operations AND(\cdot), OR($+$) and NOT(\sim) defined in conventional binary logic are extended to three values 0, 1 and 1/2 such as

$$\begin{aligned} x \cdot y &= \min(x, y), \\ x + y &= \max(x, y) \text{ and} \\ \sim x &= 1 - x \end{aligned}$$

, where $x, y \in V_3 = \{0, 1/2, 1\}$. In the B-ternary logic, as in the conventional binary logic, the commutative, associative, absorption, idempotent, distributive and DeMorgan's laws are satisfied. Yet, the B-ternary logic is characterized by the fact that (complementary laws): $\sim x \cdot x = 0$, $\sim x + x = 1$ do not hold. A B-ternary logic function is a ternary function represented by the logical formula which is obtained by applying the logical operations \cdot , $+$ and \sim , a finite number of times, to the variables x_1, \dots, x_n , where x_i takes a truth value in the three values 0, 1/2 and 1. Every B-ternary logic function

can be expanded into a disjunctive form (sum of products form) and into a conjunctive form (product of sums form). However, since the complementary laws do not hold, the forms may contain some terms in which a variable and its negation exist in pair simultaneously as a factor. We call a variable x_i or its negation $\sim x_i$ as a literal. A simple product (sum) term is defined to be the product (sum) of some literals in which a variable and its negation simultaneously does not exist in pair as a factor. A complementary product (sum) term is defined to be the product (sum) of some literals in which, at least, one pair of a variable and its negation occurs simultaneously as a factor. A complementary product (sum) term is called a complementary minterm (maxterm) if it contains each of the variables as factors. A B-ternary logic function is said to be represented by a B-ternary disjunctive (conjunctive) form if it consists of the sum (product) of simple product (sum) terms and/or complementary product (sum) terms. Every B-ternary logic function can always be represented by the B-ternary disjunctive and conjunctive forms but, in general, may have several such forms. Every complementary product (sum) term can be expanded into sum (product) of complementary minterms (maxterms). A B-ternary logic function F is said to be represented by the B-ternary canonical disjunctive (conjunctive) form if it is represented by the B-ternary disjunctive (conjunctive) form

$$F = \alpha_1 + \dots + \alpha_n \quad (F = \alpha_1 \cdot \dots \cdot \alpha_n)$$

, where α_i is a simple product (sum) term or a complementary minterm (maxterm) and $\alpha_i \not\leq \alpha_j$ for all i and j ($i \neq j$). Any given B-ternary logic function has one corresponding B-ternary canonical disjunctive (conjunctive) form which is determined uniquely.

A partially ordered relation \geq , which designates some kinds of ambiguity, is defined on the set of $V_3 = \{0, 1/2, 1\}$ and V_3^n as follows: (Fig.1)

[Definition 6] $1/2 \geq 0$, $1/2 \geq 1$ and $a_i \geq a_j$, where $a_i, a_j \in V_3$. Let $a=(a_1, \dots, a_n)$, $b=(b_1, \dots, b_n)$ be elements of V_3^n . Then, $a \geq b$ iff $a_i \geq b_i$ for all i .

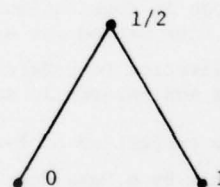


Fig.1: A Partially Ordered Relation \geq

Let F be a B-ternary logic function and a be an element of the domain V_3^n of F . Then, a^* designates the set of all elements a' of V_3^n ($V_3 = \{0, 1\}$) such as $a \geq a'$. And $F(a^*)$ designates the set of the values which $F(a')$ takes, where a' is an element of a^* .

[Theorem 1] A ternary function F is a B-ternary logic function iff

(1) If $a \in V_2^n$, then $F(a) \in V_2$,

(2) If $a > b$, then $F(a) > F(b)$.

[Corollary 1] If F is a B-ternary logic function,

(1) $F(a) = 1/2$, if $F(a^*) = \{0, 1\}$,

(2) if $F(a) = 0$, then $F(a^*) = \{0\}$,

(3) if $F(a) = 1$, then $F(a^*) = \{1\}$.

[Corollary 2] If a B-ternary disjunctive (conjunctive) form F is composed only of simple product (sum) terms, then

$F(a) = 0$ (1) iff $F(a^*) = \{0\}$ ($\{1\}$).

[Definition 7] An element $a = (a_1, \dots, a_n)$ of V_3^n corresponds to a simple product (sum) term a and vice versa iff

$a_i = 1$ (0) iff x_i exists in a ,

$a_i = 0$ (1) iff \bar{x}_i exists in a ,

$a_i = 1/2$ iff x_i and \bar{x}_i do not exist in a .

[Definition 8] An element $a = (a_1, \dots, a_n)$ of $V_3^n - V_2^n$ corresponds to a complementary minterm (maxterm) a iff

$a_i = 1$ (0) iff x_i exists and \bar{x}_i does not in a ,

$a_i = 0$ (1) iff \bar{x}_i exists and x_i does not in a ,

$a_i = 1/2$ iff x_i and \bar{x}_i exist in a .

For any pair of elements a and b of V_3^n , we can always find the least upper bound (i.e., the supremum) $c = a \cup b$ with respect to the partially ordered relation \geq . The greatest lower bound (i.e., the infimum) $c = a \cap b$ is defined as the greatest element c satisfying $a > c$ and $b > c$ if such an element c exists, and we write as $a \cap b = \phi$ if it does not exist.

[Corollary 3] Let a be a simple product (sum) term corresponding to an element a of V_3^n . Then,

(1) $a(b) = 1$ (0) iff $a > b$ iff $a(b^*) = \{1\}$ ($\{0\}$),

(2) $a(b) = 1/2$ iff $a(b^*) = \{0, 1\}$,

(3) $a(b) = 0$ (1) iff $a \cap b = \phi$ iff $a(b^*) = \{0\}$ ($\{1\}$).

[Corollary 4] Let a be a complementary minterm (maxterm) corresponding to an element a of $V_3^n - V_2^n$. Then,

(1) $a(b) = 1/2$ iff $b > a$,

(2) $a(b) = 0$ (1) iff $\bar{b} > a$,

(3) $a(b^*) = \{0\}$ ($\{1\}$) for all elements b of V_3^n .

[Corollary 5] If a B-ternary disjunctive (conjunctive) form F consists only of complementary product (sum) terms, then $F(a) = 0$ (1) for all elements a of V_2^n .

[Corollary 6] If a complementary minterm (maxterm) a exists in a B-ternary canonical disjunctive (conjunctive) form F , then $F(a) = 1/2$ holds for a corresponding to a .

4. Static Hazard Detection by B-ternary Logic

Henceforth, let F denote the B-ternary logic function realized by a combinational switching circuit F . An input in steady state is symbolized by an element $a = (a_1, \dots, a_n)$ or $b = (b_1, \dots, b_n)$ ($a_i, b_i \in V_2$) of V_2^n and an input in transient state by an element $c = (c_1, \dots, c_n)$ ($c_i \in V_3$) of V_3^n . Here, $c_i = 1/2$ means that the input variable x_i is changing from

0 to 1 or from 1 to 0. If $c_i \in V_2$, then it means that x_i does not change. The subject of this paper cen-

ters on switching circuits composed of gate-type AND, OR and NOT elements and it is assumed that each such elements does not contain any logic hazard; however, as will be shown in chapter 6, the methodology in this paper is applicable to nets constructed by any gate-type elements, even if the elements of the nets may contain any logic hazards. [Theorem 2] For F and any element c of V_3^n , $F(c) = 1/2$

holds iff the output of F may change as a result of the simultaneous change in the 1/2 input variables in c .

(Proof) The output of AND, OR and NOT elements are 1/2 iff the outputs may change when 1/2 input variables change simultaneously. Since a combinational switching circuit is constructed by AND, OR and NOT elements, the theorem is proved. (Q.E.D.)

Let a and b be two steady state inputs. Then, $c = a \cup b$ indicates a changing input state from a to b or from b to a , that is, a V_3^n element whose chang-

ing variables are 1/2. Therefore, by Theorem 2, various kinds of static hazards defined in chapter 2 can be formalized by using the B-ternary logic.

[Theorem 3] F contains a static hazard caused by an input change $a \rightarrow b$ iff

(1) $F(a) = F(b)$,

(2) $F(a \cup b) = 1/2$.

(Proof) Since inputs a and b are elements of V_2^n ,

we obtain $f(a) = F(a) = F(b) = f(b)$. As shown above, only the changing variables are 1/2 in $a \cup b$. Hence, the theorem is shown by Definition 1 and Theorem 2. (Q.E.D.)

[Theorem 4] If F contains a static hazard for an input change $a \rightarrow b$, then, F also contains a static hazard for an input change $b \rightarrow a$.

(Proof) It is evident from $a \cup b = b \cup a$. (Q.E.D.)

According to the above theorem, we need not distinguish between input change $a \rightarrow b$ and $b \rightarrow a$ with regard to static hazards. Hereafter, we write an input change as $a \leftrightarrow b$.

[Corollary 7] F contains a 0 (1) static hazard for an input change $a \leftrightarrow b$ iff

(1) $F(a) = F(b) = 0$ (1),

(2) $F(a \cup b) = 1/2$.

(Proof) It is evident from Definition 2 and Theorem 3. (Q.E.D.)

[Corollary 8] F contains a logic hazard for an input change $a \leftrightarrow b$ iff

(1) $F(a \cup b) = 1/2$,

(2) $F((a \cup b)^*) \neq \{0, 1\}$.

(Proof) $(a \cup b)^*$ is the set consisting of all elements of V_2^n derived from replacing all 1/2's in

$a \cup b$ by 0 or 1. Therefore, it expresses the set of a, b and all steady state inputs occurring during the transition $a \leftrightarrow b$. $F((a \cup b)^*) \neq \{0, 1\}$ means that $F((a \cup b)^*) = \{0\}$ or $F((a \cup b)^*) = \{1\}$, that is, $F(a) = F(b)$. Hence, we obtain the theorem by Definition 3 and Theorem 3. (Q.E.D.)

[Corollary 9] F contains a function hazards for an input change $a \leftrightarrow b$ iff

(1) $F(a) = F(b)$,

(2) $F((a \cup b)^*) = \{0, 1\}$.

(Proof) Since $F((a \cup b)^*) = \{0, 1\}$, $F((a \cup b)) = 1/2$ (1) of Corollary 1) and there exist in $(a \cup b)^*$ elements

a' and a'' satisfying $F(a')=0$ and $F(a'')=1$. In consequence, the theorem is shown by $F(a) \neq F(a')$ or $F(a) \neq F(a'')$ (Definition 4). (Q.E.D.)

A logic (function) hazard is called the logic (function) 0 (1) hazard if it is a static 0 (1) hazard.

[Theorem 5] If F does not contain a static hazard for an input change $a \leftrightarrow b$ and $F(a)=F(b)$, then, does not contain a static hazard for any input change $a' \leftrightarrow b'$ such as $aUb > a'Ub'$.

(Proof) If a static hazard is not contained in F , $F(aUb) \neq 1/2$ holds; that is, $F(aUb)=0$ or $F(aUb)=1$. If $F(aUb)=0$, then, $F(aUb) > F(a'Ub')=0$ because of $aUb > a'Ub'$ (Theorem 1). Hence, a static hazard is not contained (Theorem 3). The similar proof is applicable for $F(aUb)=1$. (Q.E.D.)

[Theorem 6] If F contains a static hazard for an input change $a \leftrightarrow b$, then F contains a static hazard for every input change $a' \leftrightarrow b'$ satisfying $F(a')=F(b')$ and $a'Ub' > aUb$.

(Proof) If a static hazard is contained, $F(aUb)=1/2$ is valid (Theorem 3). Since $a'Ub' > aUb$, we obtain $F(a'Ub') > F(aUb)$, that is, $F(a'Ub')=1/2$ (Theorem 1). Consequently, a static hazard is also contained for an input change $a' \leftrightarrow b'$ in F . (Q.E.D.)

We can detect easily by Theorem 3 whether a static hazard is contained or not in F for an input change $a \leftrightarrow b$, and we can further discern by Corollary 7 whether it is a static 0 hazard or a static 1 hazard. Although the detection of whether logic hazards or function hazards is possible by Corollary 8 and Corollary 9, the detection is difficult because we need to examine the value of F for all elements of $(aUb)^*$.

5. Static Hazard Detection and Identification through Disjunctive or Conjunctive Forms

For a combinational switching circuit F , there exists the logical formula Ψ corresponding to F . The logical formula Ψ represents a binary logic function f and a B-ternary logic function F . This chapter deals with the detection and the identification of various kinds of static hazards by deriving the disjunctive or conjunctive form from the logical formula Ψ .

[Theorem 7] Let Ψ be the formula corresponding to F . And let Ψ' be any formula obtained from Ψ by applying the laws covered by the B-ternary logic, and F' be the combinational switching circuit corresponding to Ψ' . Then, for an input change $a \leftrightarrow b$, F contains a static hazard iff F' contains a static hazard.

(Proof) Even if a law covered by the B-ternary logic is applied to a logical formula, the B-ternary logic function represented by the formula is the same. Yet, the existence of a static hazard is determined by the B-ternary logic function realized by the circuit (Theorem 3). Then, we obtain this theorem. (Q.E.D.)

By the above theorem we can see that every law held by the B-ternary logic is a transformation that preserves static hazards. On the other hand, every logic formula can be expanded into the B-ternary disjunctive (conjunctive) form. Therefore, it is sufficient to show the detection and identification methods of static hazards involved in the above two forms. Henceforth, let the B-ternary disjunctive (conjunctive) form corresponding to F represent

the B-ternary disjunctive (conjunctive) form of the logical formula Ψ corresponding to F .

[Theorem 8] If the B-ternary disjunctive (conjunctive) form corresponding to F consists only of simple product (sum) terms, then, all static 0 (1) hazards in F are function hazards.

(Proof) If a static 0 (1) hazard is contained in F for an input change $a \leftrightarrow b$, then, $F(a)=F(b)=0$ (1) and $F(aUb)=1/2$ are satisfied (Corollary 7). According to Corollary 2, if F is composed only of simple product (sum) terms, then, $F(aUb)=1/2$ is equivalent to either $F((aUb)^*)=\{1\}$ ($\{0\}$) or $F((aUb)^*)=\{0,1\}$. The former contradicts to $F(a)=F(b)=0$ (1); then, we arrive at $F((aUb)^*)=\{0,1\}$. Therefore, it is a function hazard (Corollary 9). (Q.E.D.)

[Theorem 9] If the B-ternary disjunctive (conjunctive) form corresponding to F is composed only of complementary product (sum) terms, then, F does not contain static 1 (0) hazards and function hazards. (Proof) If F consists only of complementary product (sum) terms, then, $F(a^*)=\{0\}$ ($\{1\}$) for all elements a of V_2^n in accordance with Corollary 5. Therefore, Corollaries 7 and 9 indicate that a static 1 (0) hazard and a function hazard are not contained in F . (Q.E.D.)

In general, the B-ternary disjunctive (conjunctive) form corresponding to F is written as follows:

$$F = F_{sd} + F_{cd} \quad (F = F_{sc} \cdot F_{cc})$$

, where $F_{sd}(F_{sc})$ is a formula which is the sum (product) of simple product (sum) terms and where $F_{cd}(F_{cc})$ is the sum (product) of complementary product (sum) terms. The following conclusions are derived from the two above-mentioned theorems.

[Theorem 10] F contains a function 0 hazard for an input change $a \leftrightarrow b$ iff

- (1) $F_{sd}(aUb)=1/2$,
- (2) $F_{sd}(a)=F_{sd}(b)=0$.

(Proof) We need not consider function hazards involved in F_{cd} because F_{cd} does not contain the

function hazards (Theorem 9). F_{sd} does not contain logic 0 hazard (Theorem 8); then, the theorem is derived by Corollary 7. (Q.E.D.)

[Theorem 11] F contains a logic 0 hazard for an input change $a \leftrightarrow b$ iff

- (1) $F_{sd}(aUb)=0$,
- (2) $F_{cd}(aUb)=1/2$.

(Proof) Because $F_{sd}(aUb)=0$ and $aUb > a, b$, we have $F_{sd}(a)=F_{sd}(b)=0$. Furthermore, since a and b are elements of V_2^n , $F_{cd}(a)=F_{cd}(b)=0$ holds (Corollary 5).

. Therefore, we obtain $F(a)=F(b)=0$. On the other hand, it follows from $F = F_{sd} + F_{cd}$ that $F(aUb)=1/2$.

Hence, a static 0 hazard is contained in F . Since this hazard is not a function 0 hazard because $F_{sd}(aUb)=0$ (Theorem 10), it is a logic 0 hazard. (Q.E.D.)

[Theorem 12] F contains a function 1 hazard for an input change $a \leftrightarrow b$ iff

- (1) $F_{sc}(aUb)=1/2$,
- (2) $F_{sc}(a)=F_{sc}(b)=1$.

The proof is omitted.

[Theorem 13] F contains a logic 1 hazard for an input change $a \leftrightarrow b$ iff

- (1) $F_{sc}(aUb)=1$,
- (2) $F_{cc}(aUb)=1/2$.

The proof is omitted.

It is easier by theorem 10 ~ 13 than by corollaries 8 and 9 to distinguish between logic hazards and function hazards.

6. The Detection of All Logic Hazards

For the detection of all static hazards contained in a given combinational switching circuit, we need to examine all inputs by using Theorem 3. We can examine them fairly efficiently, if we use properties discussed up to now. This chapter shows a simple method to detect logic hazards algebraically.

[Theorem 14] If F contains a logic hazard for an input change $a \leftrightarrow b$, then, F contains a logic hazard for any input change $a' \leftrightarrow b'$ satisfying $aUb = a'Ub'$. (Proof) If F contains a logic hazard for an input change $a \leftrightarrow b$, then, $F(aUb)=1/2$ and $F((aUb)^*) \neq \{0,1\}$ hold from Corollary 8. Therefore, if $aUb = a'Ub'$, a logic hazard is also contained for an input change $a' \leftrightarrow b'$. (Q.E.D.)

The above theorem shows that a logic hazard can be represented by an element c of $V_3^n - V_2^n$. Hereafter, we represent the existence of a logic hazard by an element c of $V_3^n - V_2^n$ rather than by a pair of changing inputs. That is, the statement " F contains a logic hazard for c " means that F contains a logic hazard for all input change $a \leftrightarrow b$ satisfying $aUb = c$. [Theorem 15] F contains a logic 0 (1) hazard for an element c iff there is a complementary minterm (maxterm) β in the B-ternary canonical disjunctive (conjunctive) form F corresponding to F satisfying $c \geq c_\beta$ and $c \wedge c_\alpha = \phi$ for all simple product (sum) term α of F , where c_β and c_α are the elements corresponding to β and α , respectively.

(Proof) If $c \wedge c_\alpha = \phi$ for every simple product (sum) term α of F , then $F(c^*) = \{0\} (\{1\})$ since $\alpha(c^*) = \{0\} (\{1\})$ from (3) of Corollary 3. Furthermore, if there exists a complementary minterm (maxterm) β in F , then, $F(c_\beta) = 1/2$ by Corollary 6. Therefore, if $c \geq c_\beta$, then $F(c) = 1/2$ from Theorem 1. That is, F contains a logic 0 (1) hazard for c (Corollaries 7 and 8). Conversely, if a logic 0 (1) hazard is contained for c , then, $F(c^*) = \{0\} (\{1\})$ and $F(c) = 1/2$. $F(c^*) = \{0\} (\{1\})$ means that $\alpha(c^*) = \{0\} (\{1\})$ for every simple product (sum) term α of F ; that is, $\alpha(c) = 0 (1)$. Hence, $c \wedge c_\alpha = \phi$. If $F(c) = 1/2$, then, there exists a complementary minterm (maxterm) β satisfying $\beta(c) = 1/2$. Hence, we arrive at $c \geq c_\beta$ from (1) of Corollary 4. (Q.E.D.)

[Corollary 10] If there is a complementary minterm (maxterm) β in the B-ternary canonical disjunctive (conjunctive) form corresponding to F , then, F contains a logic 0 (1) hazard for the element c_β corresponding to β .

(Proof) If there is a complementary minterm (maxterm) β in the B-ternary canonical disjunctive (conjunctive) form, then, $c_\beta \wedge c_\alpha = \phi$ for every simple product (sum) term α of F . This is because, if $c_\beta \wedge c_\alpha \neq \phi$, then, $\alpha \geq \beta$; that is, β is omitted by α .

This is contradictory to the hypothesis that F is the B-ternary canonical disjunctive (conjunctive) form. Hence, we acquire the theorem by Theorem 15. (Q.E.D.)

By obtaining the B-ternary canonical disjunctive (conjunctive) form corresponding to F through Theorem 15, we can detect all logic hazards contained in F .

The B-ternary canonical disjunctive (conjunctive) form of a B-ternary logic function, in general, written as

$$F = F_{sd} + F_{md} \quad (F = F_{sc} \cdot F_{mc}) \quad (1)$$

, where $F_{sd}(F_{sc})$ is the sum (product) of simple product (sum) terms and where $F_{md}(F_{mc})$ is the sum (product) of complementary minterms (maxterms). On the other hand, every B-ternary logic function represents a binary logic function f , provided that the variables take one of the two values $\{0,1\}$. For any given B-ternary logic function, we denote by $F_{pd}(F_{pc})$ the B-ternary logic function represented by the logic formula composed of the sum (product) of all prime implicants* (implicates*) of f corresponding to F . Then, we can show that the B-ternary canonical conjunctive (disjunctive) form of $F_{sd}(F_{sc})$ is written as

$$F_{sd} = F_{pc} \cdot F_{mc} \quad (F_{sc} = F_{pd} + F_{md})$$

Therefore, every B-ternary logic function F can be represented from (1) as follows:

$$F = F_{pc} \cdot F_{mc} + F_{md} \quad (2)$$

$$= (F_{pd} + F_{md}) \cdot F_{mc} \quad (3)$$

, and these representation are determined uniquely for F .

[Lemma] Let c be an element of V_3^n and F be a B-ternary logic function. Then, $c \wedge c_\alpha = \phi$ for every simple product (sum) term α of $F_{sd}(F_{sc})$ iff $c \geq c_\beta$ for a prime implicate (implicant) β in $F_{pc}(F_{pd})$.

(Proof) If $c \wedge c_\alpha = \phi$ for every simple product (sum) term α of $F_{sd}(F_{sc})$, then, $F(c^*) = \{0\} (\{1\})$. Since the value of $F(a)$ is equivalent to $F_{pc}(a) (F_{pd}(a))$ for every a of V_2^n , $F_{pc}(c^*) = \{0\} (F_{pd}(c^*) = \{1\})$ holds. Hence, by (1) of Corollary 3, there exists a prime implicate (implicant) β in $F_{pc}(F_{pd})$ and $c \geq c_\beta$. The converse is shown in similar manner. (Q.E.D.) [Theorem 16] F contains a logic 0 (1) hazard for c iff there are a prime implicate (implicant) α in $F_{pc}(F_{pd})$ and a complementary minterm (maxterm) β in $F_{md}(F_{mc})$ such as $c \geq c_\beta$ and $c \wedge c_\alpha = \phi$.

(Proof) It is shown by Theorem 15 and Lemma. (Q.E.D.)

For a combinational switching circuit F , the term $F_{pd}(F_{pc})$ is determined uniquely and is independent from circuit construction. We can detect

*) A simple product (sum) term α is said to be a prime implicant (implicate) of f iff $\alpha \leq f$ ($\alpha \geq f$) and there does not exist a simple product (sum) term β satisfying $\alpha < \beta \leq f$ ($\alpha > \beta \geq f$).

function hazards by $F_{pd}^{(F)pc}$ which represents all steady states of the circuit. On the other hand, the terms F_{md} and F_{mc} vary according to circuit construction and represent the transient states corresponding to logic hazards. The forms (2) and (3) represent a B-ternary logic function by separating the transient states corresponding to logic hazards from the steady states. From Theorem 15 and 16 all logic hazards in a combinational switching circuit are detectable algebraically, that is, detectable by operating the logic formula corresponding to the circuit. Conversely by applying Theorem 16 we can construct a combinational switching circuit which contains specific logic hazards. Although this paper has considered switching circuits consisting of AND, OR and NOT elements which do not contain logic hazards, the methodology described here is applicable to the circuits composed of other gate-type elements even if the elements contain logic hazards. For example, if an element does not contain logic hazards, then, let the element correspond to the B-ternary logic function represented by the sum (product) of all prime implicants (implicates) of the binary logic function realized by the element. And if the element contains specific logic hazards, then, let it correspond to the B-ternary logic function obtained by the formula (2) or (3).

7. Dynamic Hazards Detectable by B-ternary Logic

As discussed hitherto, the B-ternary logic has full description ability about the steady states and the transient states corresponding to the static hazards of combinational switching circuits. But it is not necessary that the B-ternary logic can always describe all transient states. For example, many spurious pulses may appear in the output by duplicating static hazards. However, the transformation of logic formula by the idempotent and absorption laws of the B-ternary logic does not fix the number of spurious pulses. That is, by the B-ternary logic we can discern whether or not a static hazard is contained, but we cannot know how many spurious pulses appear. Similarly, since the absorption and distributive laws are not transformations which preserve dynamic hazards, we cannot use the B-ternary logic to know about dynamic hazards, in general. But, as will be mentioned below, there exist some dynamic hazards detectable by the B-ternary logic.

[Theorem 17] F contains a dynamic hazard for an input change $a \leftrightarrow b$ if

- (1) $F(a) \neq F(b)$,
- (2) F contains a static hazard for an input change $a' \leftrightarrow b'$ such as $aUb > a'Ub'$.

(Proof) Let $F(a)=0$ and $F(b)=1$ (can be proved similarly for $F(a)=1$ and $F(b)=0$) and suppose a static 0 hazard is contained in F . Then, during the input change $a \leftrightarrow b$, the route $a \leftrightarrow a' \leftrightarrow b' \leftrightarrow b$ clearly exists, where $aUb > a'Ub'$, $F(a)=F(a')=F(b')=0$ and where a' and b' are elements of V_2^n (a' and a may coincident).

. Since a static 0 hazard is contained for the input change $a' \leftrightarrow b'$, a dynamic hazard is contained for the input change $a \leftrightarrow b$. A similar proof follows a static 1 hazard. (Q.E.D.)

[Corollary 11] F contains a dynamic hazard for an input change $a \leftrightarrow b$ if

- (1) $F(a) \neq F(b)$,
 - (2) there is a complementary minterm (maxterm) β such as $aUb > c$ in the B-ternary canonical disjunctive (conjunctive) form corresponding to F .
- (Proof) If there is a complementary minterm (maxterm) β , F contains a logic 0 (1) hazard for c (Corollary 10). Then, we obtain Corollary 11 from Theorem 17. (Q.E.D.)
- [Theorem 18] If there is a complementary minterm (maxterm) β in the B-ternary canonical disjunctive (conjunctive) form F corresponding to F , then, F contains a hazard for every input change $a \leftrightarrow b$ such as $aUb > c$.
- (Proof) Let $F(a)=F(b)$. If there is a complementary minterm (maxterm) β in F , then, $F(c)=1/2$ (Corollary 6). If $aUb > c$, then $F(aUb)=1/2$ by Theorem 1. Therefore, from Theorem 3 a static hazard is contained in F . Let $F(a) \neq F(b)$. Then, a dynamic hazard is contained in F by Corollary 11. Therefore, a hazard is contained in either case. (Q.E.D.)

8. Conclusion

This paper has shown that the question whether or not a combinational switching circuit contains a static hazard is determined totally in the scope of the B-ternary logic theory. Particularly, using the canonical forms of B-ternary logic functions realized by the circuits, we have demonstrated the detection and identification methods of various kinds of static hazards as well as an algebraic detecting method of logic hazards. Furthermore, we have pointed out that there are some dynamic hazards detectable by the B-ternary logic.

The methodology described here is applicable to any combinational switching circuit whatever the building elements may be.

The results obtained in this paper give an answer to the synthesis problem on the construction of a combinational switching circuit which contains specified logic hazards.

Acknowledgements

We would like to thank pro. Motinori Goto of Meiji University, Dr. Yasuo Komamiya, Head of the Electronic Devices Division of the Electrotechnical Laboratory, and Dr. Tadashi Nagata, Chief of the Information Control Research Group of ETL, for their continued encouragement.

References

- (1) D.A. Huffman: "The design and use of hazard-free switching net works", J.ACM, 4,1, pp.47 (Jan.1957)
- (2) E.J. McCluskey: "Transients in combinational logic circuits", in Redundancy Technique for Computing System, Spartan Book, Washington D.C., pp.9 (1962)
- (3) S.H. Unger: "Hazards and delays in asynchronous sequential switching circuits", IRE Trans. Circuit Theory, Vol. CT-6, pp.12 (March 1959)
- (4) M. Yoeli and S. Rinon: "Application of ternary algebra to the static hazards", J.ACM, 11,1, pp. 84 (Jan. 1964)
- (5) E.B. Eichelberger: "Hazard detection in combinational and sequential switching circuits", IBM J, 9, 2, pp.90 (March 1965)
- (6) J.G. Bredeson and P.T. Hulina: "Elimination of static and dynamic hazards for multiple input changes in combinational switching circuits", Inf. and Cont. Vol.20, No.2 (March 1972)

- (7) L.L.Mete, S.Das and H.Y.H.Chung: "A logic hazard detection and elimination method", Inf. and Cont. Vol.26, pp.351 (1974)
- (8) J.Beister: "A unified approach to combinational hazards", IEEE Trans. on Comp. Vol.C-23, No.6, pp.566 (June 1974)
- (9) Sugino, Inagaki and Fukumura: "A method of detecting static hazards caused by multi-input-variable changes using ternary algebra", Jour.I.E.C.E., Japan, 50, 6, pp.997 (June 1967); available in English in E.C.J., same date, pp.17.
- (10) Fukumura, Inagaki and Kohmura: "Minimization of hazard-free switching units", Information processing, Japan, Vol.8, No.4 (July 1967)
- (11) M.Mukaidono: "A formalization and a detection of hazards in combinational circuits by means of a ternary logic", Paper of Technical Group on Electronic Computers, I.E.C.E., Japan EC72-28 (Nov. 1972)
- (12) M.Goto: "Application of logical mathematics to the theory of relay networks", Jour. IEE Japan, Vol.59 (April 1949)
- (13) M.Mukaidono: "On the B-ternary logical function", (in Japanese), Trans.I.E.C.E., Japan, 55-D,6, pp.335 (June 1972)**
- (14) M.Mukaidono: "On some properties of fuzzy logic", (in Japanese), Trans.I.E.C.E., Japan, 58-D,3, pp.150 (Mar. 1975)**
- (15) M.Mukaidono: "An algebraic structure of fuzzy logical functions and its minimal and irredundant form", (in Japanese), Trans.I.E.C.E., Japan, 58-D,12, pp.748 (Dec. 1975)**
- (16) A.Kandel: "Application of fuzzy logic to the detection of static hazards in combinational switching systems", International Journal of Computer and Information Sciences, Vol.3, No.2 (1974)

** available in English in Systems·Computers·Controls (Scripta Publishing Co.); same date.

POSSIBILITY THEORY: AS A MEANS FOR MODELING COMPUTER SECURITY AND PROTECTION

David Rine
Information Science Program

Western Illinois University
Macomb, IL 61455

The paper studies correlations between computer security specifications and protection techniques in order to find ways by which architectural features of a computer system corresponding to security specifications may be designed or evaluated. The formal basis of the study begins with two basic models introduced in sections 2.1 and 2.2 of the paper. Study objectives include developing a set of models based on a theory of possibilities for "measuring" strengths and weaknesses of "correlations" between security specifications and protection features.

1. Introduction and Review of Some Previous Models.

1.1 Early Protection Mechanisms.

In the late 1940's and early 1950's, early computers were machine-language programmed by single users with programs being processed one at a time. Users could protect confidential programs and data against unauthorized access by simply taking the programs and data home (in the form of notes, punched cards, or tapes) after each use.

By the late 1950's and early 1960's, with the introduction of multiuser programming and operating systems, programs and data began to fall into two classes, either application programs which utilize user libraries, or privileged programs which could only be processed by the executive operating system. Hence, computer system protection schemes were of two levels and used devices such as switches and keys to isolate application programs/data from privileged programs/data.

In addition to privileged programs/data, it was soon discovered that some application programs and data demanded a greater degree of protection. Computer system protection schemes using devices such as protection rings were employed by users to give these more sensitive application programs/data a greater degree of security.

By the late 1960's and early 1970's, with more levels of software separating the user-nonprogrammer from the operating systems and with more diverse and complex applications and query programming security requirements, it was discovered that the protection mechanisms afforded by operating systems did not often match well, or enforce properly, the security requirements of the user community.

This problem still exists. It is, therefore, of primary importance to study the correlations between computer security policies (rules, requirements) and protection mechanisms (features, techniques) in order to exhibit and analyze those architectural features and programming techniques of a computer system corresponding to various security policies (15). This is what we propose to research.

In the past, for most data management systems it was not clear what security policy a particular protection mechanism enforced via access control. Recent research (13) has led to a more careful model(s) of logical security policies, denoted herein by I_1, \dots, I_s , and logical protection mechanisms, M_1, \dots, M_p , (as opposed to physical security policies and physical protection mechanisms which are designed to prevent unauthorized access by bypassing the system (1, 3, 16)). These newer models attempt to bridge the differences between security and protection, particularly with the concept of soundness* and the concept of completeness**. For example, the surveillance protection mechanism is sounder than and as complete as the high water mark protection mechanism (14); this will be explained later in section 2.1. This paper discusses some security policies and protection mechanisms of data base systems using the model described in section 2.1 (below, page 2), and attempts to offer a general means for deciding whether or not some of these protection mechanisms are sound.

*Associated with a protection mechanism $M(Q, I)$ which enforces a specific security policy I for a given program Q . The word "soundness" was first used in (6, 7). See, also, Appendix A herein.

**Defined in order to determine whether or not one protection mechanism $M(Q, I)$, particularly those which are first sound, out-performs the others $M(Q, I)$. That is, one wishes to judge the degree of completeness, the degree to which all possible penetration holes, but only those holes, are plugged.

Early works were primarily concerned with protection mechanisms and did not formally differentiate them from security policies. For example, Graham (4) proposed an outline of a model for ring protection mechanisms like those used in MULTICS-like operating systems. In this ring protection model protection of data and programs is provided during the execution or run-time of a given procedure, as opposed to protection by checking of authorized access by the procedure at input/output or source code translation time.

1.2 Early Security Principles and The Confinement Problem.

Although there was an implied security policy called a "security principle," in Graham's model, the definition of the security principle was never spelled out.

Later, there followed a first attempt to precisely define and formalize the concept of a security policy by means of a security matrix model.

For example, Conway and others (2) proposed a model for security policies based on an access security matrix representation of privacy decisions. This (2) is one of the early references in which processing cost is discussed. Thus, it was concluded that cost of protection is proportional to two variables: (a) the cost of checking security at translation time, which is data independent; and (b) the cost of checking security at execution time, which is data dependent.

In the paper (2) security policies were denoted "information privacy decisions," and protection mechanisms were denoted "information security actions". A need for "soundness" of protection mechanisms, which we discuss in section 2.1, might possibly be derived from the following quote: "In most information systems to date, the privacy decision has not been made explicit and security has not been deliberately implemented" (2).

More recent data base management systems have, for the most part, chosen to implement access control and protection mechanisms at the translation step because it is easier to do so. For example, INGRES (5) has implemented access control in its query language QUEL in the following way: the "RESTRICT" QUEL command is appended to a user's qualification by the AND operator. An IBM query language similar to QUEL called SEQUEL, a query language for System-R, does translation time access control checking. From (2) one may conclude that some efficiency is gained if the appending is done at query translation time and if one assumes complete data independence. However, there is still the matter of the Trojan Horse Problem of application program access (the Trojan Horse Problem does not normally apply to query language access) and the Observability Postulate Assumption.

2. Current Research and Proposed Developments

2.1 Basic Model

In this section a basic model is described which associates with a user or program both a security policy and an associated protection mechanism. This model is similar to one proposed by Jones and Lipton (8) but includes some additional components worthy of further investigation. The model is introduced as consisting of the following components:

Basic Model

- OB) Objects to be protected, logical and physical.
- OP) Operations* on objects, logical and physical.
- EN) Environment:** subset of OP x OB, logical and physical.
- MT) Enforcement: protection mechanisms - software and hardware, as a function of time.
- N) Environment Modification: Logical and physical data base migration.
- G) Migration: $EN_1 \rightarrow EN_2$ or $EN_1/\text{restriction} \rightarrow EN_2/\text{restriction}$.
Program (user or subject-program)
 $Q: D_1x \dots xD_k \rightarrow E_1x \dots xE_n$
 D_i : range of i th input
 E_j : range of j th output
 $X_1, \dots, X_k \in OB$
 Y_1, \dots, Y_n output variables
 $Q(X_1, \dots, X_k) = (Y_1, \dots, Y_n) =$
 $(Q_1(X_1, \dots, X_k), \dots, Q_n(X_1, \dots, X_k)).$

Security policy I_j , $1 \leq j \leq n$, for Y_j is a collection of subsets $\{X_{j1}, \dots, X_{jm}\}$: $\forall X_{js} \exists X_p \in \{X_1, \dots, X_k\} \supset X_{js} \subseteq X_p$. Security policies $\{I_1, \dots, I_n\}$ is the security policy I of Q .

Given Q and I introduce protection mechanism $M(Q, I): D_1x \dots xD_k \rightarrow E_1x \dots xE_n$ where $F(M)$ violation notices, $E'_1 = E_1 \cup F(M)$, $E_1 \cap F(M) = \emptyset$, and $M_1(Q, I)(X_1, \dots, X_k) = Q_1(X_1, \dots, X_k) = Y_1$ or $M_1(Q, I)(X_1, \dots, X_k) \in F(M)$. From the motivation by Conway, et. al. (2) and Lampson (9) one introduces "soundness". Consider $Q, I, M(Q, I)$. $M(Q, I)$ sound if $\forall j, Y_j, I_j = \{X_{j1}, \dots, X_{jm}\} \exists M'(Q, I)/\text{restricted } M(Q, I): D_{j1}x \dots xD_{jm} \rightarrow$

*Of which subjects' processes are composed; e.g. authorized accesses are operations.

**Associated with each user is an environment. Dynamic alteration or modification of data bases should preserve security.

$$E_1'x \dots xE_n' \exists M_j(Q, I) = M_j'(Q, I) = Y_j.$$

I.e. $M_j'(Q, I)$ only gives information derived from I_j authorized information; $M_j(Q, I)$ could give information derived from either I_j or not I_j if there is a leak as illustrated by Figure 1 below.

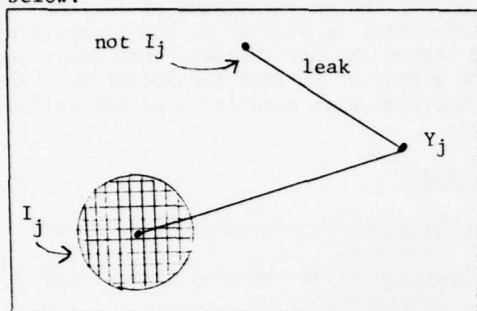


Figure 1. Sound Protection Mechanisms: unsound mechanism, indicating a possible leak to output variable Y_j .

While it is true that a sound protection mechanism enforces a security policy for a user program, it is also true that it may over-enforce the policy thereby generating unnecessary violation notices. Consider the following graphic remark.

Remark: Suppose $\exists M(Q, I) \ni \forall i M_i(Q, I) (X_1, \dots, X_k) = C \in F(M)$ fixed. Then $M(Q, I)$ is useless and gives too many violation notices. Hence, there is a dichotomy between over-enforcement and soundness.

To address this dichotomy, the concept of "Completeness" is introduced in order to use lattice operations to build up, by the meet and join operations from sound protection mechanisms, a protection mechanism which neither over-enforces nor allows leaks.

Informally, completeness in the general sense implies that all those and only those requests Q that are security sensitive must pass through the internal protection mechanisms $M(Q, I)$ before being honored.

As a compromise one may only be able to get a new protection mechanism which is sound and which does not over-enforce, i.e. generate as many violation notices, as much as the old protection mechanism. One introduces "completeness" as follows:

Completeness (lattice concepts)

Let $M^1(Q, I), M^2(Q, I)$. Then, $M^1(Q, I) \geq M^2(Q, I)$ whenever $M^1(Q, I)(a) \neq Q(a)$ implies $M^2(Q, I)(a) \neq Q(a)$.
Let $M^1(A, I), M^2(Q, I)$ sound. Define $M = M^1 \cup M^2$ join $\ni \forall a$

$$M(a) = Q(a), \text{ if } \exists i \in 1, 2 \ni M^i(a) = Q(a), \\ M^1(a), \text{ otherwise.}$$

Conclusion: M is sound and $\forall i M \geq M^i$.

Pf. Induction.

A total access control mechanism $M(I)$ is said to be complete when every attempt Q to access information during the progress of a computation is validated. This is, generally, impossible since it is not known in advance what all Q would be. However, completeness does compare the extent to which the security requirements I , if complied with, provide assurance that the information system is not vulnerable to penetration (i.e. compares which mechanism is better).

Most protection mechanisms are not completely defined for all Q or are made incomplete as the result of software or hardware failures. Hence, there is a need for added deterrents to attacks. Individual users' knowledge that their activities are being monitored may deter them from conducting attacks. Philosophically, a surveillance protection mechanism is complete when each and every attempt to access information during a computation is either monitored, detected, or recorded, this being true only for information of which auditing was required.

A more recent discussion of security policies versus protection mechanisms is provided by McCauley (10). In McCauley's thesis the "Conceptual Model" is introduced to formalize security policies as opposed to protection mechanisms. The conceptual model consists of the following components:

Access types: program operations a , the program type identifiers;
Extended Logical Data Base (ELDB): all triples (u, a, d) , u is a user and d is a data element;
protection specifications; two-valued functions $p: S \rightarrow \{\text{permit}, \text{deny}\}$, $S \subseteq \text{ELDB}$;
protection pattern: P set of all p 's;
consistent P : P is a function;
authority item: user;
capability list: inaccessible, blocked, or open;
ordering " \leq ": $P \leq Q$, P and Q protection patterns, and P a restriction of Q ;
specification ordering: $p \leq q$;
 $\text{glb}(p, q)$: greatest lower bound of p, q with respect to " \leq ", used to eliminate inconsistencies.

In the above model members of the ELDB can be viewed as instances of HYDRA (17) pairs (s, f) , s protected object and f access function. Also, the ordering \leq of protection patterns is similar to the ordering $M_1 \leq M_2$ of protection mechanisms previously defined in this paper.

If two-valued functions $p: S \rightarrow \{\text{permit}, \text{deny}\}$, $q: s \rightarrow \{\text{permit}, \text{deny}\}$ are protection specifications, then define $A_p = \{d: p(u, a, d) = \text{permit}\}$,

$A_q = \{d: q(u,a,d) = \text{permit}\}$. If $x \notin A_p \cap A_q$, then it is a point of inconsistency when p, q enforce I_j . To eliminate the inconsistent point, form and use $\text{glb}(p,q): S \rightarrow \{\text{permit}, \text{deny}\}$ as follows:

$$\text{glb}(p,q)(d) = \begin{cases} \text{permit, if } p(d) = q(d) \\ \text{deny, otherwise.} \end{cases}$$

See Figure 2.

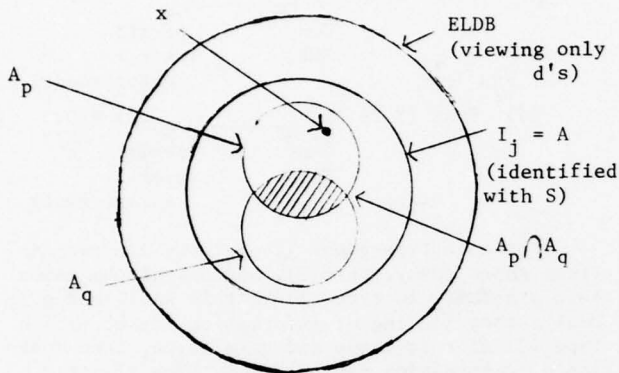


Figure 2. Protection Analysis by Lattice Models: Inconsistent point x for two sound protection mechanisms p, q .

It is also easy to show that in general the protection specification (10) and the security matrix (2) are equivalent to the security policy I of this paper. For if $c(u,d)$ is a security matrix where u : user, d : data element, then define

$$\tau(u,d) = \begin{cases} \text{permit iff } c(u,d) = 1 \\ \text{deny iff } c(u,d) = 0 \end{cases}$$

Then, let $M(Q, I_u): D \rightarrow E'$ and $M_u(Q, I_u)$:

$$D_1 \times \dots \times D_k \rightarrow E_u \cup F(M).$$

Let $d \in D_1 \times \dots \times D_k$, $u: I_u$, $a: Q$; and, then $M_u^1(Q, I_u)(d) = e_u$ or f .

Now define

$$\rho_u(Q, I_u)(d) = \begin{cases} \text{permit iff } M_u(Q, I_u)(d) = e_u, e_u \in E_u \\ \text{deny iff } M_u(Q, I_u)(d) = f, f \in F(M)_a \\ \text{violation notice.} \end{cases}$$

Finally, define

$$\lambda(Q, I_u)(d) = \begin{cases} 1 \text{ or permit iff } d \in I_u \\ 0 \text{ or deny iff } d \notin I_u. \end{cases}$$

The two-valued functions above can easily be extended to multiple-valued functions. (Appendix B).

2.2 Further Study.

2.2.1 Correlations Between Policies and Mechanisms.

In the previous section the high water mark protection mechanism $M_h(Q, I; G, >, G_0)$ was introduced as a model for protection features in operating systems such as the ADEPT-50 (16). Recall that if Q is a flowchart user program and I is a security policy, the $(G, >)$ is a linearly ordered set with grading $>$ and $G: \{X_1, \dots, X_k, Y, \dots\} \rightarrow G$ is a function from the variables of Q to G . An assumption is that there exists $p \in G$ such that $\forall v \in I, w \notin I$, then $G_0(w) > p \geq G_0(v)$. The following are example interpretations:

- (1) G is a set of classifications (military system of clearances) such as "top secret", "secret", "confidential", "public", ..., which are ordered such that top secret $>$ secret $>$ confidential $>$ public ...;
- (2) initially all the variables in a security policy I are given some classification equal to or lower than p , and all other variables are given some classification higher than p ;
- (3) $G = \{0, 1\} = \{\text{deny}, \text{permit}\}$.

Recall the following connection from the previous section between security policies and security matrices (2):

$$\lambda(Q, I_u)(d) = \begin{cases} 0, d \text{ denied to } u \\ 1, d \text{ permitted to } u. \end{cases}$$

This connection could be extended to multiple-values (16) as follows:

$$\lambda(Q, I_u)(d) = \begin{cases} 0, d \text{ denied to } u \\ 1, d \text{ usually denied to } u \\ 2, d \text{ sometimes permitted to } u \\ 3, d \text{ usually permitted to } u \\ 4, d \text{ permitted to } u. \end{cases}$$

0 corresponds to the highest cost (penalty) to break the protection mechanism, 1 to the next highest cost (penalty), and 4 to the lowest cost (no penalty). For the protection mechanism the values 0, ..., 4 could also be interpreted as protection ring numbers (4).

Military classification schemes seem to be completely binary; either access is permitted or it is denied. However, in formulating security policies, even in the military, human use of policy statements is subject to vague interpretations; and managers use such terms as "sometimes permitted" in trying to describe security policies.

Another way of viewing what takes place in passing from the security policy in the first

example (1) above to its protection mechanism is to observe that "confidential" is a fuzzy set $\ast(19)$ having an initially vague meaning. Likewise, are top secret, secret, public, denied, usually denied, sometimes permitted, usually permitted, and permitted fuzzy sets.

Continuing the notion of fuzzy sets, suppose that $U = \{0,1,2,3,4,5,6,7,8,9,10\}$ is a universe of discourse consisting of all security classes or protection ring numbers. Let N be a variable taking values in U and u the generic element of U . Then $N = 2$ signifies that N is assigned the ring number 2. Suppose that the "grade" of membership in fuzzy set "confidential" is approximately .8, $\ast E(G, >)$. Next, from (18) let "Poss" denote the possibility** distribution (function).

Continuing the concept of possibility distributions, consider the "degree of possibility that 'dataset is 2' given the proposition 'dataset is confidential'". This can be denoted as follows:

Poss (dataset is 2 | dataset is confidential) = .8, where 'confidential' is a fuzzy subset of U .

In the area of management, administrators often must express security policies by using imprecise terminology. The following are examples:

dataset A is confidential.
dataset B is denied access.
dataset C is restricted but dataset D is permitted access.

*fuzzy set F : U =universe of discourse (e.g. $[0,1]$); mapping $\mu_F: U \rightarrow [0,1]$; F a subset of U , and a "linguistic label" of the subset, and characterized by μ_F ; $F = \int_U \mu_F(u)/u$, infinite F ; $F = \mu_1 u_1 + \dots + \mu_n u_n$, finite F . $\mu_F(u)$ is a

"membership grade", interpreted as the "compatibility of u with the concept labelled F ".

**possibility: F fuzzy subset of U ; X a variable over U ; $F = R(X)$, fuzzy restriction on X , such that $X=u$: $\mu_F(u)$, where $\mu_F(u)$ is the "degree to which the constraint represented by F is satisfied when u is assigned to X "; possibility distribution $\pi_X = R(X) = F$; and possibility distribution function, possibility that $X=u$, $\pi_X(u) = \mu_F(u)$.

Possibility distribution statements such as Poss (A is 2 | A is confidential) = .8 and Poss (B is 1 | B is denied) = .2 illustrate that Poss is a connection between policies and enforcements, where for example .2 and .8 are meanings (result of an evaluation) of enforcement.

A general possibility distribution statement is Poss (X is r | X is F) = g , "X is F " is a policy, F fuzzy subset, "X is r " is a desired enforcement, and $g \in (G, >)$ is an evaluation of enforcement.

Consider the following types of possibility distribution statements:

(1) Poss (X is 0 |) = 0.

deny (leak) permit (OK)

(2) Poss (X is 10 |) = 1.

deny (leak) permit (OK)

(3) Poss (X is 0 |) = 1.

deny (OK) permit (over enforcement)

(4) Poss (X is 10 |) = 0.

deny (OK) permit (over enforcement)

For each type above (1-4) there are two extreme cases (deny, permit), and one of the cases is a problem. In type (1) if r is small and g is small, then leaking of information may occur; in type (2) if r is large and g is large, then leaking of information may occur; in type (3) if r is small and g is large, then over-enforcement may occur; and, in type (4) if r is large and g is small, then overenforcement may occur.

Thus from the previous section 2.1 (p. 2) of the paper, types (1) and (2) illustrate a lack of soundness for the protection mechanism; whereas types (3) and (4) illustrate a lack of completeness, i.e. the protection mechanisms for types (3) and (4) are less than optimal with respect to the lattice ordering defined in the last section for sound protection mechanisms in particular.

Let us abbreviate Poss (X is r | X is F) to Poss (r | F).

One can graph the possibility curve on a pair of U , G axes as illustrated in Figure 3 in order to gain a better view of the optimal point (complete protection mechanism with no leaking of information from the data base). As in type (3) and type (2) possibility statements, set the g value grade evaluation level to, for example, 1. As the Poss possibility curve's graph moves from Poss (0 | permit) of type (3) to Poss (10 | deny) of type (2) over enforcement of an associated protection mechanism decreases as one moves to the optimal point. As the Poss possibility curve's graph moves past the optimal point (assuming there is but one), leaking of information from the data base increases.

Dually, as is illustrated in Figure 4 and as for type (1) and type (4), set the g value grade evaluation level to, for example 0. As the Poss curve moves from Poss (0 | deny) to Poss (10 | permit) leaking decreases to the optimal point, and over enforcement increases past the optimal point.

It then becomes of interest to find a means by which the optimal point can be located.

If over enforcement is not as serious as leaking of information, then users may be satisfied in choosing a point to the left (right) of the optimal point in figure 3 (figure 4), thereby adding an additional constraint into the graph.

The ability to answer questions relating to

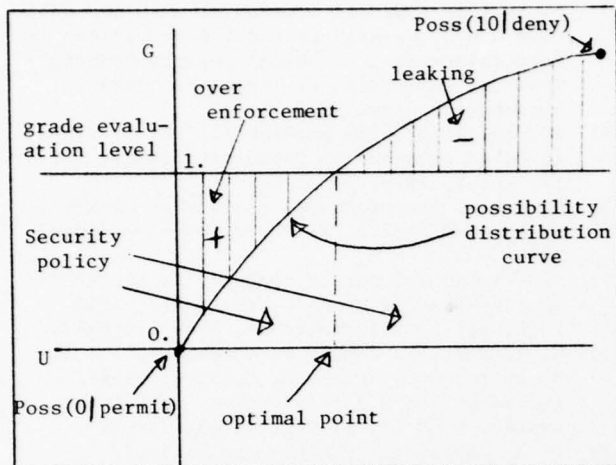


Figure 3. Security and Protection Analysis by Possibility Models: Graph of Possibility Curves $Poss(0|permit)$ type (3) to $Poss(10|deny)$ type (2) and optimal point representing complete protection mechanism with no leakage of information from the data base.

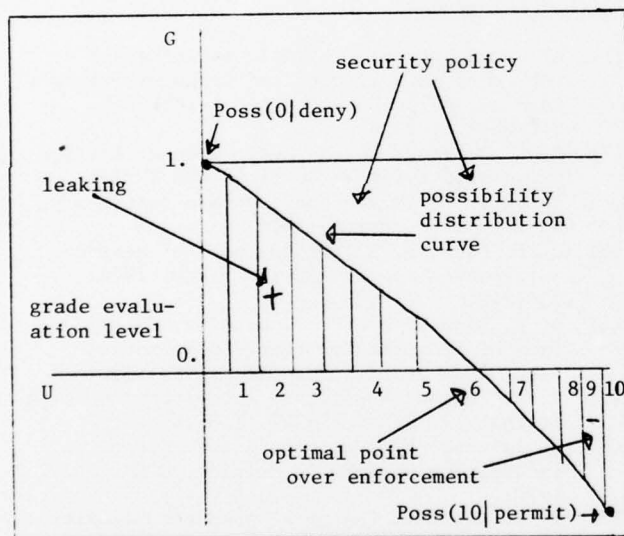


Figure 4: Security and Protection Analysis by Possibility Models: Graph of Possibility Curves $Poss(0|deny)$ type (1) to $Poss(10|permit)$ type (4) and optimal point representing complete protection mechanism with no leakage of information from the data base.

security policies for the authorized accessing and extracting of implicit or explicit information from data that is stored in a data base is being discussed. A theory of possibility (18) may help lead to a model for illustrating how this can be done with a (somewhat) complete and sound protection mechanism.

In conclusion we have indicated how important it is to study the correlation(s) between

security policies (rules, requirements, specifications) and protection mechanisms (features, techniques) in order to define and analyze specific architectural features of a computer system corresponding to specific security policies. We have seen that it may be advantageous to measure strengths or weaknesses of such correlations through possibilities. Important tools may include the usage of possibility curves, such as those previously illustrated.

A report in the same general area of study using fuzzy sets has recently been written by Michelman and Hoffman (11). However, the study reported herein is different, and has different objectives, from their study in that we are concerned with the correlations between security specifications and protection mechanisms, with the intent of changing one or the other if their correlation is low. Michelman and Hoffman (11) are, on the other hand, concerned with evaluations of the security specifications themselves. A general relation between the two studies can be described in the following way: Let T , O , F be their (11) sets of threats, objects, and security features respectively. Let F be equivalent to our given security specifications, described herein, and let PM be sets of corresponding protection mechanisms. In Figure 5 below let C be a set of correlation values, such as possibility values described earlier. We are studying $C(F, PM)$.

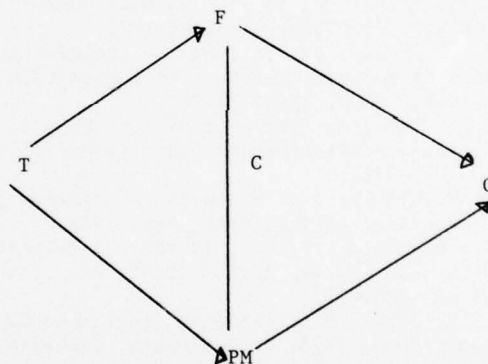


Figure 5: A diagram relating threats, objects, security specifications, protection mechanisms, and correlations.

If I is a security policy (see section 2.1), then $M(Q, I)$ is a corresponding protection mechanism which attempts to flag illegal requests by enforcement. Let V stand for a profile or set of values representing what a user or administrator "thinks" the security policy is, possibly being a vague or imprecise idea. On the other hand I is a precisely stated security policy. In the Figure 6 below, one may choose to evaluate relations e between $F=V$ and $F=I$, or one may choose to investigate correlations c between $F=I$ and $PM=M(Q, I)$.

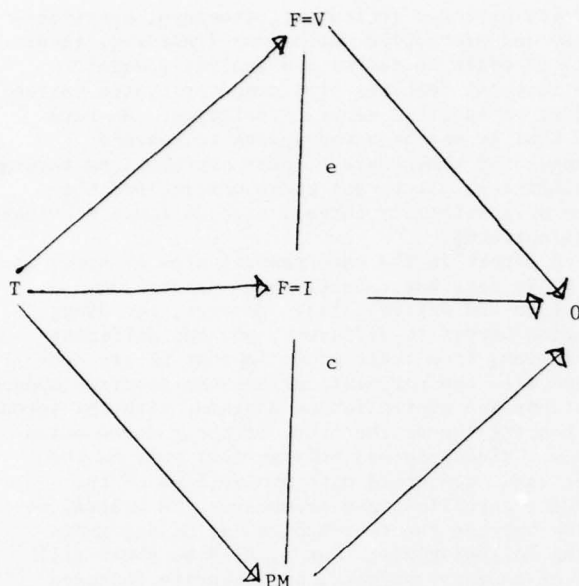


Figure 6. A diagram relating threats, objects, security specifications, and protection mechanisms.

References.

- (1) C. W. Beardsley, Is your computer secure?, I.E.E.E. SPECTRUM, January, 1972.
- (2) R. W. Conway, et. al., On the implementation of security measures in information systems, CACM, April, 1972.
- (3) C. J. Date, An Introduction to Database Systems, Addison-Wesley, 1975 (especially pp. 295-296).
- (4) R. M. Graham, Protection in an information processing utility, CACM, May, 1968.
- (5) G. D. Held, et. al., INGRES - a relational data base system, AFIPS - Conf. Proc., vol. 44 and NCC, 1975.
- (6) A. K. Jones, Protection in Programmed Systems, June, 1973, Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh.
- (7) A. K. Jones and W. A. Wulf, Towards the design of secure systems, Software - Practice and Experience, 5 (1975), 321-336.
- (8) A. K. Jones and R. J. Lipton, The enforcement of security policies for computation, Proc. Fifth Symp. on Operating Systems Principles, Operating Syst. Rev. (ACM SIGOPS Newsletter), Vol. 9, 5 (November, 1975), 197-206.
- (9) B. W. Lampson, A note on the confinement problem, CACM, October, 1973.
- (10) E. McCauley, A Model for Data Secure Systems, 1975-76, Ph.D. Thesis, The Ohio State University, Columbus.
- (11) E. H. Michelman and L. J. Hoffman, SECURATE: a security evaluation and analysis system using fuzzy metrics, memorandum no. UCB/ERL M77/36, Electronics Research Lab., University of California, Berkeley, July, 1977.
- (12) D. C. Rine, Multiple-valued logic and flowcharting, Australian Computer Science Journal, vol. 7, no. 2, July, 1975.

- (13) D. C. Rine, Simple models for security policies and associated protection mechanisms in data management, technical report, Computer Science, University of Texas, San Antonio, December-January, 1976-77.
- (14) G. Schlageter, The problem of lock by value in large data bases, Computer Journal, vol. 19, no. 1, 1976.
- (15) L. Smith, Architectures for secure computing systems, MTR-2722, MITRE Corporation, June, 1974.
- (16) C. Weissman, Security controls in the ADEPT-50 time-sharing system, Proc. AFIPS 1969 FJCC, Vol. 35, 10 (October, 1969), 119-133.
- (17) W. A. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, R. Pollack, HYDRA: the kernel of a multiprocessor operating system, CACM 17, 6(June, 1974), 337-345.
- (18) L. A. Zadeh, Fuzzy sets as a basis for a theory of possibility, memorandum no. UCB/ERL M77/12, Electronics Research Lab., University of California, Berkeley, February.
- (19) L. A. Zadeh, K. S. Fu, K. Tanaka, and M. Shimura, eds., Fuzzy sets and Their Applications to Cognitive and Decision Processes, Academic Press, New York, 1975.

Related References.

- (1) E. Cohen, et. al., HYDRA: basic Kernel reference manual, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, November 4, 1976.
- (2) D. E. Denning, On the derivation of lattice structured information flow, CSD TR 180, Computer Sciences Dept., Purdue University, W. Lafayette, March, 1976.
- (3) D. E. Denning, A lattice model of secure information flow, CACM 19, 5(May, 1976), 236-243.
- (4) D. E. Denning and P. J. Denning, Certification of programs for secure information flow, CACM 20, 7(July, 1977), 504-513.
- (5) J. S. Fenton, Memoryless subsystems, Computer Journal 17, 2(May, 1974), 143-147.
- (6) J. Saltzer, Protection and the control of information sharing in Multics, CACM, July, 1974.
- (7) K. S. Shankar, The total computer security problem: an overview, I.E.E.E. COMPUTER, June, 1977.
- (8) S. Winkler and L. Danner, Data security in the computer communication environment, I.E.E.E. COMPUTER, February, 1974.

Appendix A.

It is helpful to be able to view the model described in section 2.1 of this paper in terms of a real commercial data base system (DBS) and associated protection mechanism. Because of its popularity, let us consider an example like IBM's IMS DBS.

Security policies may be implemented via IMS protection mechanisms in the following ways:

- (1) segment sensitivity (SENSEG),
- (2) processing options (PROCOPT=),
- (3) data encryption before storage, and
- (4) terminal and password security.

AD-A055 763

ILLINOIS INST OF TECH CHICAGO DEPT OF COMPUTER SCIENCE F/G 12/1
PROCEEDINGS OF THE INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED L--ETC(U)
MAY 78 A S WOJCIK, R SWARTWOUT N00014-78-G-0019

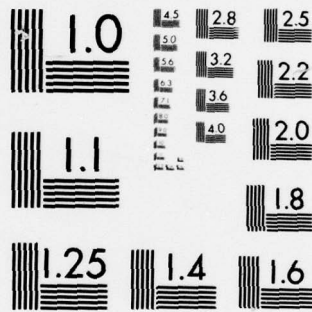
UNCLASSIFIED

NL

4 OF 4
AD
A055763



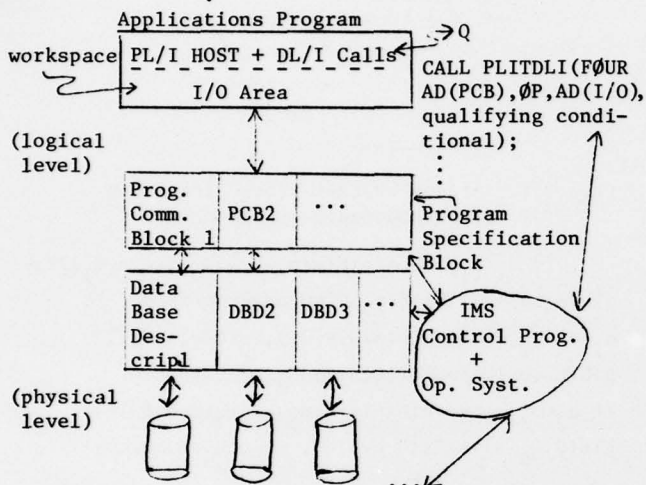
END
DATE
FILMED
8 -78
DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Figure 1 illustrates the basic architecture of an IMS System. At the physical storage level data encryption provides data access protection. At the communications level data encryption and terminal protection provides data access protection. At the applications program level password protection provides data access protection. At the DL/I calls level from the host language record segment sensitivity and authorized processing options provide data access protection.

In terms of our model Q is a DL/I call, an X_j^s is a program communication block (PCB), an X_p^s is a data base description (DBD),
DLITPLI:PROCEDURE(PCB1,PCB2,...,PCBN)OPTIONS(MAIN);



IMS Data Bases

Figure 1. Architecture of IMS System.

and, indeed, $\forall s \exists p \ni X_j \in X_p$, i.e. the segments of each PCB reside in some DBD. Moreover, the set of PCB's assigned legally to a given applications group, $\{X_j, \dots, X_m\}$, describes the complete restricted view of the data base (data submodel), $\{X_1, \dots, X_n\}$, for which they have authorized access; and, therefore, it is indeed the realization of their security policy I_j . Hence, for all programming groups who may wish to issue a given type of DL/I call, Q: CALL PLITDLI(FOUR,*,OP,I/O Area, qualifier), security policies $\{I_1, \dots, I_n\}$ is, indeed, the security policy I of Q.

Y_j is precisely the I/O Area to which users programs' authorized PCB's $\{X_j, \dots, X_m\}$ may return data from a DL/I call without raising an error flag, i.e. violation notice. $Q(X_{j_1}, \dots, X_{j_m})=Y_j$.

We raise the question as to whether or not IMS under SENSEG and PROCOPT is sound, as a protection mechanism, given a DL/I call type, Q, and a designated security policy, I, over access to all PCB's via DBD's. That is, does IMS really enforce a policy formalized in SENSEG/PROCOPT notation? Consider the employee data base described in Figure 2.

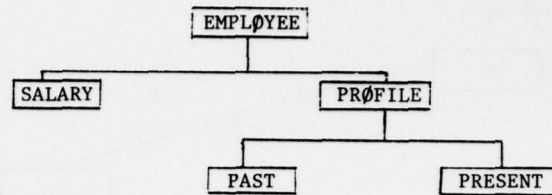


Figure 2. Employee IMS DB, physical data base PDB1.

Let EMPLOYEE, PROFILE, And PRESENT be sensitive segments authorized to PCB1; and, let SALARY, PAST be non-sensitive segments unauthorized to PCB1, and protected in theory by PROCOPT=K. Therefore, the restricted view of the data base is as in Figure 3.

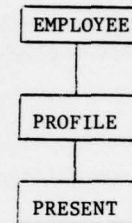


Figure 3. Employee IMS logical data base PCB1.

Suppose that a query DL/I call, Q, requests data about PRESENT information regarding an EMPLOYEE.

Then, internally, IMS must follow from the root (EMPLOYEE), segment occurrences according to the hierarchical sequence (tree preorder traversal) from EMPLOYEE occurrences, through SALARY, PROFILE, PAST, and PRESENT occurrences, until the desired PRESENT segment occurrence is obtained, as in Figure 4.

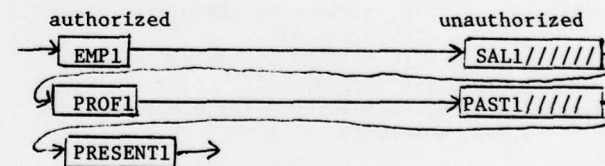


Figure 4. Internal data structure traversal.

Thus, the accessing program must pass through physical records which are unauthorized to the user.

Should a system program mechanism be developed which could, while the access program is processing, leak physical records which are being passed through, then the protection mechanism would not be sound, as depicted in Figure 5.

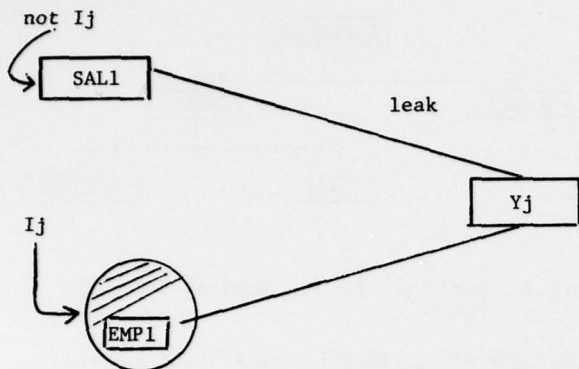


Figure 5. A possible leak to I/O Area Yj.

Thus, there may not exist $M^1(Q, I)$ such that $M_j(Q, I) = M^1_j(Q, I) = Y_j$.

A better protection mechanism would involve a data structural arrangement prohibiting pass throughs.

Appendix B.

Access types: program operations a , the program type identifiers.

ELDB: triples (u, a, d) , u : user, a : operation, d : data element.

Protection Specifications; L chain lattice and special Post algebra, $L(\leq) = \{e, e_2, \dots, e_{n-1}\}$, n -valued functions $p: S \rightarrow L$, $S \subseteq \text{ELDB}$.

Protection pattern: P , set of all p 's.

Consistent P : P is a function on ELDB, i.e. no inconsistent points.

Ordering " \leq ": $P \leq Q$, via the L ordering, P and Q protection patterns, and P a restriction of Q .

Specification ordering: $p \leq q$, via the L ordering, p and q protection specification, i.e. $\forall (u, a, d)$ $p(u, a, d) \leq q(u, a, d)$, meaning that p is more restrictive than q .

$\text{glb}(p, q): S \rightarrow L$, defined by $\text{glb}(p, q)(u, a, d) = \text{glb}(p(u, a, d), q(u, a, d))$.

$\text{lub}(p, q): S \rightarrow L$, defined by $\text{lub}(p, q)(u, a, d) = \text{lub}(p(u, a, d), q(u, a, d))$.

$D_i(p): S \rightarrow L$, defined by $D_i(p)(u, a, d) = D_i(p(u, a, d))$, monotonic.

Example:

Let $L(\leq) = N(\leq) = \{0, \dots, n-1\}$ ring numbers of ring protection(4), using the natural ordering. Then protection specifications are ring protection mechanisms. All protection access brackets (n_1, n_2) , such that $q(u, a, d) = n_1 \leq n_2 = p(u, a, d)$ for authorized (u, a, d) , are described by the function $\text{glb}(p, D_i(a))$. In particular if $i=0$, 0: deny, $n-1$: permit, then $\text{glb}(p, D_0(q))$ describes the usual permit/deny Boolean situation. See Figure 1. Let $p(u, a, d) = n_2$ if $d \leq n_2$; 0 otherwise. Let $q(u, a, d) = n_1$ if $d \leq n_1$; 0 otherwise.

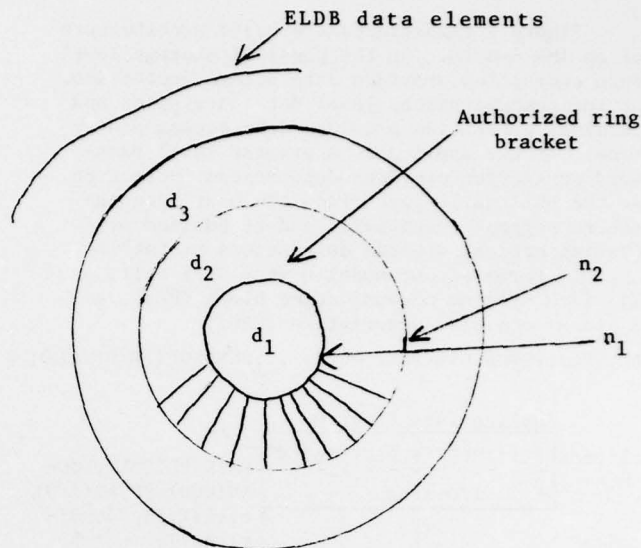


Figure 1. Access bracket protection using a Post/monotonic system.

If $d_1 < n_1$, then $\text{glb}(p(u, a, d), D_0(q(u, a, d_1))) = \text{glb}(n_2, D_0(n_1)) = \text{glb}(n_2, 0) = 0$, unauthorized. If $n_1 \leq d_2 \leq n_2$, then $\text{glb}(p(u, a, d_2), D_0(q(u, a, d_2))) = \text{glb}(n_2, D_0(0)) = \text{glb}(n_2, n-1) = n_2$, authorized. If $d_3 > n_2$, then $\text{glb}(p(u, a, d_3), D_0(q(u, a, d_3))) = \text{glb}(0, D_0(0)) = \text{glb}(0, n-1) = 0$, unauthorized.

It is now interesting to note that Post monotonic systems can be used in describing specific protection mechanisms. For example, referring to Figure 2, assume ring numbers n_1, n_2 , data access bracket (n_1, n_2) , $n_1 \leq n_2$, data item d in segment with ring number n_1 implies WRITE(to) and READ(from), data item d in segment with bracket (n_1, n_2) implies READ(from) only, and data item d in segment otherwise implies unauthorized access.

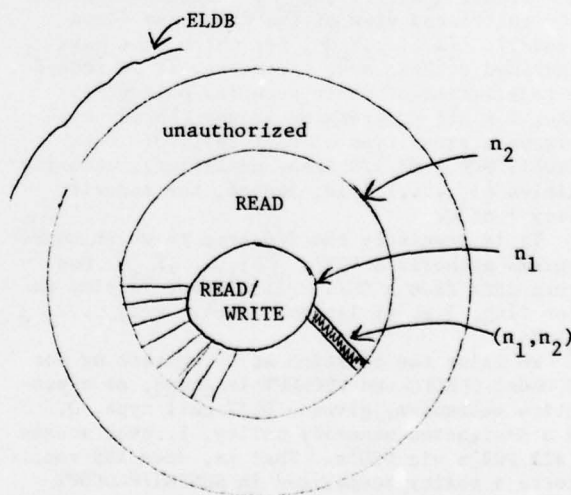


Figure 2. Protection rings for $M(Q, I)$.

Then, for a given user program Q with READ's to DB and WRITE'S from DB, and I differentiating READ authority from WRITE/READ authority for a user associated with $i \in N$,

$$M(Q, I) = (Dn_1(q) \rightarrow \text{WRITE/READ}, \\ \text{glb}(p, D_0(q)) \rightarrow \text{READ}, n-1 \rightarrow \text{ABEND}).$$

Thus, $M(Q, I)$ can be represented as a Post monotonic system conditional expression with $n-1$ interpreted as TRUE. Specifically, with $u: Q(i), i \in N$, $a: \text{WRITE or READ}$, $d: \text{segment with } (n_1, n_2)$ and $d \in \text{DB}$, then $(u, a, d) \in \text{ELDB}$ implies

$$M(Q, I)(u, a, d) = (Dn_1(q(u, a, d)) = n-1 \rightarrow \text{WRITE/READ}, \\ \text{glb}(p(u, a, d), D_0(q(u, a, d))) = n-1 \rightarrow \text{READ},$$

$n-1 = n-1 \rightarrow \text{ABEND}$), where ABEND signals an abnormal ending or abnormal ending with threat monitoring routine.

As a second example, and referring to Figure 3, assume ring numbers n_1, n_2, n_3 , program access bracket (n_1, n_2, n_3) , $n_1 \leq n_2 \leq n_3$, code C in segment with ring number n_1 implies process code, code C in segment with bracket (n_1, n_2) implies process code, code C in segment with bracket (n_2, n_3) implies call the gate keeper routine to see if entry point is in the gate list, and code C in segment otherwise implies unauthorized subroutine (procedure) call (attempted access to illegal code).

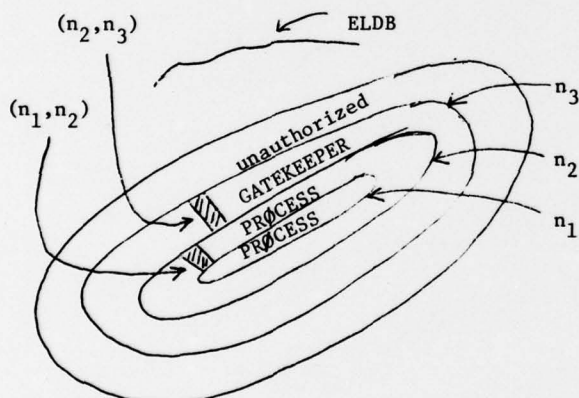


Figure 3. Protection rings for $M(Q, I)$.

Then, for a given user program Q which attempts to call (service) program P at entry point E, code in DB, and I differentiating process authority from qualified process authority at only certain entry points for a user associated with $i \in N$,

$$M(Q, I) = (Dn_1(q) \rightarrow \text{PROCESS PRG.}, \\ \text{glb}(p, D_0(p)) \rightarrow \text{PROCESS PRG.}, \\ \text{glb}(r, D_0(p)) \rightarrow \text{GATEKEEPER ROUTINE}, \\ n-1 \rightarrow \text{ABEND}), \text{ where}$$

$$r(u, a, d) = \begin{cases} n_3, & (u, a, d) \in S, \text{ i.e. } (u, a, d): i \leq n_3, \\ 0, & \text{otherwise.} \end{cases}$$

Specifically, with $u: Q(i)$ calls P at E, $i \in N$, $a: C$ of P, $d: \text{segment with } (n_1, n_2, n_3)$ and $d \in \text{DB}$, then $(u, a, d) \in \text{ELDB}$ implies

$$M(Q, I)(u, a, d) = (Dn_1(q(u, a, d)) = n-1 \rightarrow \\ \text{PROCESS PRG.}, \text{ glb}(p(u, a, d), \\ D_0(q(u, a, d))) = n-1 \rightarrow \text{PROCESS PRG.}, \\ \text{glb}(r(u, a, d), D_0(p(u, a, d))) = n-1 \rightarrow \\ \text{GATEKEEPER ROUTINE}, n-1 = n-1 \rightarrow \text{ABEND}).$$

If $M(Q, I)$ fails, then P is said to be unconfined (9).

Appendix C.

Let F be a fuzzy subset of universe of discourse $U = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, X a variable over U such that $F = R(X)$ a fuzzy restriction on X, and let $\pi_X = R(X) = F$ a possibility distribution. Then $\mu_F(i) = \mu_i \in [0, 1]$, and $F = \mu_1 u_1 + \dots + \mu_{10} u_{10}$ where μ_i is the compatibility of u_i with the concept labelled F. If $X = u_i$, then μ_i is the degree to which the constraint represented by F is satisfied when u_i is assigned to X.

If $u=2$ and $F = \text{nearby secure}$, linguistic label, then μ_2 is the degree to which X is nearby secure. Thus, $\mu_F(u) = \pi_X(u) = \mu_F(2) = \mu_2$ expresses how possible it is that X is nearby secure. .2?, .8?, 0?, 1? And, $\pi_X = F = \text{nearby secure} = \mu_2 u_1 + \dots + \mu_{10} u_{10}$.

In terms of the notation used in this paper, if $F: \text{nearby secure}$ and $\pi_X = F$, then $\pi_X(r) = \mu_F(r) = g = \text{Poss}(X \text{ is } r \mid X \text{ is } F) = \text{Poss}(r \mid F)$.

If $F_1 = \text{Permit} = \sum_U \mu_{F_1}(u) / u = \mu_1^1 u_1 + \dots + \mu_{10}^1 u_{10}$ and $F_2 = \text{Deny} = \sum_U \mu_{F_2}(u) / u = \mu_1^2 u_1 + \dots + \mu_{10}^2 u_{10}$, fuzzy sets, then it is conjectured that $\exists u^1 \in U$ such that $\pi_X(u^1) = \mu_{\text{Permit}}(u^1) = 1$, and $\exists u^{11} \in U$ such that $\mu_X(u^{11}) = \mu_{\text{Permit}}(u^{11}) = 0$, $\mu_{\text{Deny}}(u^{11}) = 1$. More over, the graph in Figure 1 depicts the possibility distribution function for F_1 .

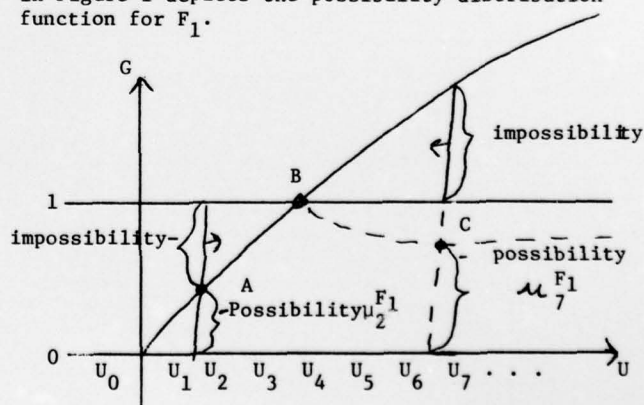


Figure 1. Possibility distribution function for F_1 .

All possibility distribution function values
are on or below the $g=1$ line. $\mu_{1/2}^F : A, \mu_1^F : B,$
 $\mu_{7/1}^F : C.$

FAULT SECURE MULTIPLE-VALUED LOGIC NETWORKS

James E. Smith

Dept. of Elect. and Comp. Eng.
University of Wisconsin-Madison

Jean Dussault

Western Electric Eng. Res. Ctr.
Princeton, New Jersey

Abstract

Fault secure multiple-valued logic networks have their outputs encoded in an error-detecting code so that assumed failures either result in no output error or in a detectable (noncode) output. In this paper we discuss a way of constructing fault secure combinational networks as well as a generalization that we define to be "strongly fault secure." The only constraint on the networks themselves is that they be made of positive unate gates and use unordered output encodings.

The fault assumptions are quite general; they include stuck-at faults as well as many other failures which seem reasonable in a multiple-valued technology. Protection is provided against both permanent and intermittent faults. Various unordered codes and their properties are discussed as is the construction of check circuits that are themselves fault secure or strongly fault secure.

I. Introduction

In theory, multiple-valued logic networks enjoy many advantages over two-valued networks [1]. Nevertheless, two-valued networks are used almost exclusively in practice, and a major reason is that two-valued logic networks are more reliable. To take one example, multiple-valued networks may have more critical noise tolerances than two-valued networks. In order for multiple-valued logic with its attendant advantages to be of practical use, the reliability problem must be solved.

There are two fundamental approaches to increasing the reliability of logic networks. The first is to simply increase the reliability of the individual components (e.g. gates). The second approach is applied at a higher level and involves interconnecting components so that reliability is increased due to the structure of the interconnection. A very simple example of the second approach is the use of triplication and voting [2]. Of course, these two approaches can be used separately, or they can be used in combination.

This paper presents an approach to the reliability problem that falls into the second

of the above categories. We discuss a class of combinational multiple-valued logic networks that have their output signals encoded in an error-detecting code. These networks are designed so that assumed failures (faults) either result in a detectable error (i.e. a noncode output) or in no error at all. This approach is based on the philosophy that the costly errors are those that go undetected. Errors can still occur using the proposed method, but they are immediately detected so that undesirable consequences can be minimized.

It appears that intermittent faults may be of great concern in multiple-valued networks due to such things as reduced noise tolerance. An additional advantage of the networks discussed here is that they are very effective at combating intermittent faults, and it is felt that the method given is one of the few practical ways of dealing with intermittent faults.

Networks that have the error-detecting property informally described above have studied for two-valued logic and are said to be "fault secure" [3]. Formal definitions of the fault secure property and a useful generalization are given and discussed in the sequel. The theory developed here is in some ways a straightforward extension of existing theory developed for the two-valued case [3-8]. However, in the area of fault modeling we depart from the obvious extension, and it is this departure that leads to what we feel are interesting and practical results.

II. Preliminaries

A. Basic Definitions

We consider multiple input, multiple output combinational logic networks which are formed as acyclic interconnections of multiple-valued gates. Let G be an m -valued logic network. Then if G has n primary input lines, the m^n vectors of length n form the input space, X , of G . The output space, Y , is similarly defined to be the set of m^p vectors of length p where G has p primary outputs. During normal, i.e. failure-free, operation G receives only a subset of X called the input code space, A , and produces a subset of Y called the output code

space, B. Members of a code space are called code words. Under faults, noncode words may be produced.

B. Post Algebras

The logic networks to be studied here are based on Post algebras [16,17]. For an m -valued system we use Post chains formed from the set of integers $[0,1,2, \dots, m-1]$ and the natural partial ordering \geq . A function or order n (i.e. n variables) possesses m^n entries in its truth table so that there exist m^{m^n} truth tables (and functions) or order n .

Post showed that the following two operators form a functionally complete set:

- 1) $f+g \triangleq \max(f,g)$
- 2) $p \rightarrow \triangleq (p+1) \bmod m$.

Another important operator, not needed for functional completeness, but equally simple to generate and helpful in constructing functions is the minimum operation:

- 3) $f \cdot g \triangleq \min(f,g)$.

With the min and max operators being the greatest lower bound and least upper bound, respectively, the lattice of truth values is a chain lattice, an example of which is shown in Figure 1a. Figures 1b and 1c show the lattices pxp and $pxpxp$ and indicate a partial ordering among m -valued vectors or vertices.

Although forming a complete set of operators with max and min functions, the cycling operation is not the only unary function of interest. For an m -valued system, there are m^m such functions. As an example, consider the 27 possible ternary functions of a single variable:

X	X^*
0	00000000011111111222222222
1	0001112220001122200011222
2	012012012012012012012012
	*** ** * ** *

Unary functions will be denoted as $X(a_0, a_1, \dots, a_{m-1})$ and will take value a_i if $X=i$, $0 \leq i < m-1$. For example $X^* = X^{(120)}$ in ternary algebra. For constructing logic networks it is desirable to limit the number of unary operators for the usual reasons associated with modularity. Due to other considerations, however, it may not always be possible to use a minimum number.

The partial ordering relation indicated in Figure 1 can be reformulated into the concept of covering.

Definition 1: A vertex $x_1 x_2 \dots x_n$ covers

a vertex $y_1 y_2 \dots y_n$, denoted as $x_1 x_2 \dots x_n \geq y_1 y_2 \dots y_n$ if $x_i \geq y_i$ for all i .

Using the max and min operators one can say $x_1 x_2 \dots x_n \geq y_1 y_2 \dots y_n$ if $\max(x_i, y_i) = x_i$ for all i , and $x_1 x_2 \dots x_n \leq y_1 y_2 \dots y_n$ iff $\min(x_i, y_i) = y_i$ for all i .

If two vertices do not satisfy in any way the binary ordering relation, they are said to be incomparable. Functions that have outputs that are ordered in essentially the same way as their arguments are called linear monotonic or unate.

Definition 2: A function $F(x_1 x_2 \dots x_n)$ is positive unate if $x_1 x_2 \dots x_n \geq y_1 y_2 \dots y_n$ implies $F(x_1 x_2 \dots x_n) \geq F(y_1 y_2 \dots y_n)$.

This notion can be readily extended to multiple output functions by requiring that each output function be positive unate.

In Section 3 we construct combinational logic networks from positive unate gates. Any such network (typically with multiple outputs) must also be positive unate.

Theorem 1: Any combinational logic network composed solely of positive unate gates must realize a positive unate function.

For generality, we do not insist on a particular set of positive unate gates, however, one might be interested in sets of gates that can implement all positive unate functions.

Definition 3: A set of positive unate operators $\{U_1, U_2, \dots, U_k\}$ is positive unate complete, or for our purposes simply unate complete, if the operators can be composed to form all positive unate functions. The use of logical constants is allowed.

Since the cycling, min, and max operators are functionally complete they can be used to construct all positive unate functions. However, the cycling operator is not positive unate, so this set of operators is not unate complete. It follows that other unary operators must be used with the max and min operators in order to get a unate complete set. In the table of single variable ternary functions, the positive unate functions are marked with an asterisk (*). Some positive unate unary functions can be obtained from others, for example $\max(X^{002}, X^{012}) = X^{(012)}$, and therefore one of the arguments is not essential. We do not address the problem of finding the smallest unate complete set(s). Nevertheless, it should be apparent that the min, max, and all positive unate unary operators are sufficient to implement all unate functions, and hence form a unate complete set of operators.

The partial ordering relation \geq can be used to define a class of codes with which we are interested.

Definition 4: An unordered code is a set of mutually incomparable vertices of similar degree.

The code {002,011,101,020,110,200} is an example of an unordered code. Specific unordered codes are discussed in section IV.

C. Fault Modeling

All failures that can occur in a network G are assumed to be modeled logically as faults. A fault is some transformation in the logic function realized by the network. By looking at the logical effects of a hardware failure, one can deal with failures within the structure of Post algebras.

For two-valued logic networks, the most common fault assumption is that failures can be modeled logically as lines "stuck-at" logical values (either 0 or 1) [9]. This stuck-at model has led to many useful results in two-valued logic. Nevertheless, we feel that a stuck-at model is inadequate for multiple-valued networks. This is primarily because a good fault model is technology dependent, and it is not all clear that the stuck-at model will model the most likely failures in a multiple-valued technology. For example, due to signal degradation in a gate (a reasonable failure mode) output voltage levels could become shifted downward. Consequently, outputs may have incorrect logic values (they are consistently too low), but are clearly not stuck. Faults of the type just described, as well as those where an upward voltage shift occurs, will be referred to as "skew faults".

In this paper we use two very general fault models. For a given network G , we define F_s to be a fault set that includes the transformation of any single gate in G so that it realizes any other single-output function. F_s is defined to be the set of single gate transformations to any positive unate function. Both F_s and F_u are very rich fault sets. F_s clearly includes all stuck-at and skew faults, and when all the gates implement positive unate functions as they do in the sequel then F_u also includes stuck-at and skew faults. These two fault models include many other failures with F_s being nearly the most general fault model possible, given that only single gates fail.

For later convenience, we define F to be the set of multiple gate transformations in G . Hence, $F_s \subseteq F$ and $F_u \subseteq F$. To denote a member of F_m involving two (or more) members of F_s , say f_1 and f_2 , we write the fault as $f_1 \cup f_2$.

The number of faults in a digital system is typically modeled as a Poisson process [10]. As a consequence, it is assumed that fault in F_s or F_u accumulate G one at a time with some time interval separating them. A critical assumption regarding this time interval is discussed in the next section. Because faults accumulate with time, we denote this behavior

by defining a fault sequence $\langle f_1, f_2, \dots, f_n \rangle$ to represent the event where f_1 occurs, followed latter by f_2 , and so forth until f_n occurs.

III. Fault Secure Network Structures

A. Definitions

We now formally define fault secure combinational networks and an interesting generalization, strongly-fault secure networks. In order to make concise definitions possible, we denote the output of the network G under input $x \in X$ and fault $f \in F_m$ as $G(x, f)$. Under the fault-free condition, the output is denoted as $G(x, \emptyset)$.

The following definition is due to D. A. Anderson [3]. It refers to the functional block G with input code space $A \subseteq X$, output code space $B \subseteq Y$, and with some assumed fault set F .

Definition 5: G is fault secure (FS) with respect to F if

$$\forall f \in F \forall a \in A \quad G(a, f) = G(a, \emptyset) \text{ or } G(a, f) \notin B.$$

If G is FS and an assumed fault occurs, then any erroneous output can be detected by simply checking for code or noncode network outputs. Hence, a fault secure network appears to offer complete protection against undetected errors caused by modeled failures. Unfortunately, this is not necessarily true. As time passes, the network may pick up a second fault, f_2 , in addition to the first, f_1 . If $f_1 \cup f_2 \in F$ then some code input may cause the output to be an undetected incorrect code word.

A partial solution to this problem is to repair the network when the first noncode output appears. If f_1 can cause a noncode output and a sufficient time passes between f_2 and f_1 for a subset of A to be applied that tests for f_1 , then this solution is satisfactory. A further problem, however, is that f_1 may be such that $\forall a \in A \quad G(a, f_1) = G(a, \emptyset)$, i.e. there is no input code word that "tests" for the presence of f_1 .

Despite the above complications a FS network is still useful from a practical standpoint if it is periodically tested offline (possibly using noncode inputs). Using this technique, an undetected error can occur only if between two off-line tests.

- 1) a fault occurs that is not tested by a code input,
- 2) a second fault occurs,
- 3) the second fault conspires with the first to give an erroneous code output.

We believe that in most situations, the probability of all three of these events occurring during a suitably chosen testing interval would be very small, and a FS network would be

effective. It would almost certainly be more reliable than an unprotected system using the same testing interval.

To develop another solution to the above problem, we first observe that G may be FS with respect to $f_1 \cup f_2$ even though $f_1 \notin F$. Then, if $f_1 \cup f_2$ results in a noncode output for some code input, the faults are detected and the network can be repaired. The following definitions (from [5]) formalize this property for fault sequences of arbitrary length.

Definition 6: For a fault sequence $\langle f_1, f_2, \dots, f_n \rangle$, $f_i \in F$, let k be the smallest integer for which there exists a code input $a \in A$ such that $G(a, \bigcup_{j=1}^k f_j) \neq G(a, \emptyset)$. If there is no such k , set $k=n$. Then G is strongly fault secure (SFS) with respect to the fault sequence if

$$\forall a \in A \text{ either } G(a, \bigcup_{j=1}^k f_j) = G(a, \emptyset) \text{ or}$$

$$G(a, \bigcup_{j=1}^k f_j) \notin B.$$

Definition 7: The network G is strongly fault secure (SFS) with respect to the fault set F if G is SFS with respect to all fault sequences whose members belong to F .

It can be shown that if a sufficient time elapses between faults so that the complete set of code inputs is applied, then the first erroneous output in response to any assumed fault (s) in a SFS network must be a noncode word. Consequently, if the above conditions are met and repairs are made whenever a fault is detected, then no undetected errors will be emitted from the network. In addition, no periodic offline testing is required; however, in many instances offline testing may be advisable to ensure that a sufficient test set (code inputs) has been applied between faults.

Two-valued SFS logic networks where k is 1 for all fault sequences have been called "totally self-checking" and have been studied extensively [3,6-8,11-14]. The typical (equivalent) definition given for these networks is:

Definition 8: A network G is totally self-checking with respect to F if

- 1) it is fault secure with respect to F and
- 2) $\forall f \in F \exists a \in A$ such that $G(a, f) \notin B$ (the self-testing property [3]).

We have chosen not to emphasize totally self-checking networks because we believe that our fault models are so general (particularly F_s) that they make the construction of totally self-checking networks very difficult. There are two arguments to support this contention.

First, in order to self-test all members

of F it is necessary that all possible input combinations be applied to all gates by members of A , and for each combination a path must be sensitized [15] to at least one output. The totally self-checking networks studied elsewhere have been rather difficult to construct, and they are based on stuck-at fault assumptions which only require some proper subset of gate input combinations be applied to each gate with the gate's output being sensitized to network outputs.

Second, the most straightforward way of constructing totally self-checking networks involves transforming a network with a non-self-tested stuck-at fault into one realizing the same function without undetectable faults. This method involves the removal of certain "redundant" parts of the network [14], i.e. the untested lines and some associated gates. For the fault models we propose here, no such transformation is apparent.

B. Main Theorems

In this section we show that any network made of positive unate gates which uses an unordered output code space must be FS with respect to F_s and SFS with respect to F_u . Consequently, a very general design method for these types of networks is to

- 1) choose some unordered output encoding;
- 2) implement the desired function using network of positive unate gates.

Using this method, the designer has a great deal of freedom both when choosing codes and designing a network. Further, later in this section we will discuss conditions under which the restriction to positive unate gates can be relaxed.

Theorem 2: Any combinational network G composed only of positive unate gates and using an unordered output code space is FS with respect to F_s .

Proof: Without loss of generality, say the fault $f \in F$ is present in G , and it affects gate g (refer to Fig. 2a). Cut the output line of g and treat it as a network input line. Then we have a new network G' with inputs x_1, x_2, \dots, x_p, g which must realize a positive unate function since it contains only positive unate gates. Apply any input $a \in A$ to G , and let $g(a, \emptyset)$ be the normal output of gate g (no fault is present). Let $g(a, f)$ be the output of g when it is faulty. Then, $g(a, \emptyset) \geq g(a, f)$ or $g(a, f) \geq g(a, \emptyset)$ (the ordering of elements of the algebra is total. If $a \circ g(a, \emptyset)$ represents the concatenation of the input word a with $g(a, \emptyset)$, then $a \circ g(a, \emptyset) \geq a \circ g(a, f)$ or $a \circ g(a, f) \geq a \circ g(a, \emptyset)$). By considering the structure of G' shown in Fig. 2b, we can deduce that

- (1) $G'(a \circ g(a, \emptyset), \emptyset) = G(a, \emptyset)$, and
- (2) $G'(a \circ g(a, f), \emptyset) = G(a, f)$.

Since G' is positive unate,

$G'(a \circ g(a, f), \emptyset) \geq G'(a \circ g(a, \emptyset), \emptyset)$
 or $G'(a \circ g(a, \emptyset), \emptyset) \geq G'(a \circ g(a, f), \emptyset)$.
 Substituting from (1) and (2);
 $G(a, f) \geq G(a, \emptyset)$
 or $G(a, \emptyset) \geq G(a, f)$.
 Consequently, either $G(a, f) = G(a, \emptyset)$ or
 $G(a, f) \setminus B$ since all members
 of B are unordered. Q.E.D.

Theorem 3: Any combinational network G composed only of positive unate gates and using an unordered output code space is SFS with respect to F_u .

Proof: For an arbitrary fault sequence $\langle f_1, f_2, \dots, f_n \rangle$, $f_i \in F_u$, let k be the smallest integer for which $\exists a \in A$ such that $G(a, \bigcup_{j=1}^k f_j) = G(a, \emptyset)$.

If there is no such k , then it follows that G is SFS with respect to the sequence by Definition 6.

If there is such a k , then $G(a, \bigcup_{j=1}^{k-1} f_j) = G(a, \emptyset)$ for all code inputs. Let G' denote the network G with the transformations of gates indicated by $k-1$

$\bigcup_{j=1}^k f_j$. Then G' is effectively made of positive unate gates and $\forall a \in A$ $G'(a, \emptyset) = G(a, \bigcup_{j=1}^{k-1} f_j)$. By

Theorem 2 G' is FS with respect to F_s and $F_u \subseteq F_s$; consequently, $\forall a \in A$ $G'(a, f_k) = G(a, \bigcup_{j=1}^k f_j) = G(a, \emptyset)$

or $G'(a, f_k) \setminus B$. Then G is SFS for the fault sequence. Since we chose an arbitrary fault sequence, the above argument must hold for all sequences. Q.E.D.

C. Implementations with Nonunate Gates

Thus far we have discussed networks made of positive-unate gates, and this has been the only restriction on network structure. We have not insisted on any particular set of gates; any unate-complete set will do. As a consequence, one could take any single-output subnetwork and call it a "gate". Since it is made of positive unate gates, it will also be positive unate. As long as its function is positive unate, any internal realization of the "gate" can be used. Hence, some nonunate gates can be interconnected to form a single-output positive unate "gate" in one of our networks.

Conceptually, the network satisfies Theorem 2 and is FS with respect to F_s , the set of all transformations of positive unate gates. It can also be seen that F_s includes all transformations of all the nonunate component gates as well. When considering F_u as in Theorem 3, some restrictions must be made on faults occurring in the nonunate component gates. In most cases, however, we feel that an adequate fault model results.

D. Interconnections of Networks

In order to be of practical use, it must be possible to interconnect FS and SFS networks in order to form larger FS and SFS networks. As before, we consider combinational networks.

In an interconnection of networks, the output code space of one network may be the input code space of other network(s). Consequently, there must not be a conflict between the use of unordered input code spaces and the restriction that networks be made of positive unate gates. Fortunately, as the following Lemma points out, unordered input codes and positive unate gates fit together quite well.

Lemma 1: Any multiple-output function whose inputs are unordered can be embedded in a positive unate function if don't care inputs are properly assigned.

The proof of the Lemma is very straightforward and has been omitted.

Another consideration when interconnecting the proposed networks is that when placed in a particular system, not all members of A may ever reach the inputs of a network. The following Lemma shows that this causes no problems.

Lemma 2: Let G be a network made of positive unate gates with an unordered output code space B and with input code space A . Then the same network G with a new input code space $A' \subset A$ is FS with respect to F_u .

Proof: Since $A' \subset A$, B' is the new output code space, and $B' \subseteq B$. Therefore, B' must be unordered, and the Lemma follows from Theorems 2 and 3.

We close this section with a theorem showing that essentially no difficulties are encountered when interconnecting the networks we propose.

Theorem 4: Given a set of networks $N = \{N_1, N_2, \dots\}$ that are each FS (SFS) with respect to F_u (according to Theorem 2 (3)), any acyclic interconnections of the networks in N forms a network that is also FS (SFS) with respect to F_u .

E. Intermittent Faults

An intermittent fault is one that does not remain permanently in a network after it first appears. After its first occurrence, we say that an intermittent fault is either 'present' or 'nonpresent'. When a member of F_s is intermittent and it is present, the network must be FS since a present intermittent fault behaves in the same way as a permanent fault. A nonpresent intermittent fault does not affect the network at all, so it is trivially FS under this condition. Consequently, a network G as described in Theorem 2 is FS

with respect to all intermittent members of F_s as well as the permanent ones.

When considering the SFS property for intermittent faults, there is some difficulty, primarily because some members of a fault sequence could be present, and others nonpresent. In short, the SFS property does not easily extend to intermittent members of F_u .

IV. Unordered Codes

In this section, we define some specific unordered codes and examine some of their properties.

A. Fixed-Weight Codes

Consider a vertex $x_1 x_2 \dots x_n$, $0 \leq x_i \leq m-1$. Then the weight of the vertex is defined to be $\sum_{i=1}^n x_i$.

Then, as their name indicates, fixed-weight codes are made up of vertices of identical weights. In pxp and ppxpx of Fig. 1b and c, each horizontal row forms a fixed-weight code space. For example, the code {002, 011, 101, 020, 110, 200} is a fixed-weight code where the weight is 2. Larger fixed-weight codes can be obtained by performing weight preserving concatenation products of smaller lattices. For the binary case, fixed-weight codes are often called k-out-of-n codes where k is the number of 1's.

In an m-valued systems, the maximum weight that a length n code word can have is $(m-1) \cdot n$. We define the fixed-weight code of weight $\lfloor \frac{(m-1) \cdot n}{2} \rfloor$ to be the half-weight code. The binary analog is the $\lfloor \frac{n}{2} \rfloor$ -out-of-n code.

Theorem 5: In an m-valued system, a length n unordered code with the maximum number of code words is the half-weight code.

The proof of Theorem 5 is a generalization of the one used to show that $\lfloor \frac{n}{2} \rfloor$ -out-of-n codes are optimal binary unordered codes [18]. As a consequence of the theorem, if one wants to minimize the number of digits used to encode information the half-weight codes will lead to such optimal encodings.

A special case of the fixed-weight codes are the "two-rail codes." The most basic two-rail code word has length 2, and these two digits, ab, have the property that $b = a^{m-1, m-2}, \dots, 1, 0$, i.e. b is the negation of a. Its code words all have weight m-1. Larger two-rail code spaces can be obtained by simple concatenation of the basic two-rail codes. For example, {0202, 0211, 0220, 1102, 1111, 1120, 2002, 2011, 2020} is the 3-valued length 4 two-rail code.

B. Berger Codes

Berger codes [19] were first constructed

for binary asymmetric channels; the following is a multiple-valued generalization. A Berger code in our generalized sense is a separable code consisting of separate information digits to which are concatenated check digits. Separable codes are useful when the extraction of encoded information is to be done most efficiently.

To satisfy the unordered property the weight of the check digits should decrease as the weight of the information digits increases. There are many ways of doing this, we consider only one useful case. A Berger code is a code consisting of information digits to which are appended the value in base m-1 of the weight of the m-1's complements of the information digits. Berger codes can be characterized by three numbers: The number of truth values, m; the number of information digits, n; and the number of check digits, k. Clearly, $k = \lceil \log_m (n \cdot (m-1) + 1) \rceil$. If $\lceil \log_m (n \cdot (m-1) + 1) \rceil = \log_m (n \cdot (m-1) + 1)$ then the code is maximal.

Example: A maximal Berger code with $m=3$, $n=4$, and $k=2$.

Info. digits	2's comp.	Weight ₁₀	Weight ₃	Codeword
0000	2222	8	22	000022
0001	2221	7	21	000121
0002	2220	6	20	000220
0010	2212	7	21	001021
:	:	:	:	:
1021	1201	4	11	102111
1022	1200	3	10	102210
1100	1122	6	20	110020
:	:	:	:	:
2220	0002	2	02	222002
2221	0001	1	01	222101
2222	0000	0	00	222200

Theorem 6: In an m-valued system, a length n separable unordered code with the maximum number of code words is the Berger code.

As with Theorem 5, a proof is a straightforward generalization of the one used for the binary case [19].

V. Check Circuits

In the networks proposed here, faults are detected by checking the network outputs for noncode words. Consequently, one must be able to construct check circuits to perform this code checking. A complication is that any such check circuit is presumably as prone to failure as the circuits being checked.

In this section, we consider checkers that are themselves fault secure or strongly fault secure. Checkers are functional blocks that in addition to satisfying the previously defined properties also possess the code disjoint property.

Definition 9: A functional block is code disjoint if it maps all noncode inputs to noncode outputs.

A checker may have a code space input consisting of a product of several codes spaces, but generally has a single and small output code space.

Fault secure check circuits are for the most part easy to construct. One can simply use a single output line with one output value indicating that the checker is examining a code word, and with the other values indicating a noncode input is being applied. That such a checker is fault secure with respect to F (or actually any member of F_m) can be easily shown. One should note, however, that after a fault occurs in the checker, it may no longer be code disjoint.

Strongly fault secure checkers, on the other hand, are much more interesting. This is because we will insist that they retain the code disjoint property for all initial fault sequences which result in no detectable output errors (referring to Definition 6, the sequences of length 1, 2, ..., k-1).

Theorem 7: In order for a checker to be SFS with respect to F_u , at least two output lines are necessary.

Proof: If only one output line is used and the line becomes stuck at a "valid" indication (this fault is in F_u) then the check circuit will not produce a noncode output for this length 1 fault sequence. However, the circuit is no longer code disjoint. Consequently, more than one output line is required. Q.E.D.

A consequence of this theorem is to define the size of the hardware, i.e. the smallest number of output lines a checker may have and still possess the code disjoint and SFS properties. Two output lines are usually sufficient, and two lines are used in the next sections where checkers for specific codes are discussed.

A. SFS Checkers for Fixed-Weight Codes

From the general discussion on checkers it is apparent that the code disjoint property imposes a partition on the set of input vertices. More precisely the set of input code words will map onto a set of output codewords. Due to the positive unateness of the checker, a noncode input to the checker that covers a codeword should produce at the checker's outputs a noncode word that covers some output codeword. A dual of this statement is also true if the covering is the other way around.

The theory and implementation of checkers for binary fixed-weight codes is well developed [4, 7, 12]. Some of the work deals with the theory of Ramsey numbers from combinatorics [7], and its extension to the multivalued case is very difficult. As an alternative, we shall pre-

sent a method to construct checkers based on the work of Carter and Schneider [4] and Anderson [12].

The checker will realize two functions; more precisely codewords will be mapped onto 2 output codewords (0, m-1) and (m-1, 0). To design a checker for a code of fixed-weight k the input digits to the checker are partitioned into two non-empty groups A and B of n_a and n_b digits. The weight of the digits in group A(B) will be labeled $w_a(w_b)$. Define the function $T(w_a \geq i)$ to have value m-1 if the argument is true and 0 otherwise. Let the 2 output functions be defined as follow:

$$f = \max_{i=0}^{\alpha} T(w_a \geq i) \cdot T(w_b \geq k-i) \quad i \text{ even}$$

$$g = \max_{i=0}^{\alpha} T(w_a \geq i) \cdot T(w_b \geq k-i) \quad i \text{ odd}$$

Where the bound α can be obtained as follow:

$$T(w_a \geq i) = 0 \quad \text{for } i > w_a$$

$$\text{and } T(w_b \geq k-i) = 0 \quad \text{for } i \geq k-w_b$$

$$\text{hence } \alpha = \min(w_a, k-w_b).$$

To complete the design method we need a technique for implementing the threshold functions. $T(w_a \geq i)$ is to be implemented. There are n_a digits in that group labeled (say) x_1, x_2, \dots, x_{n_a} . First enumerate all the n_a -tuples that have weight w_a . For each variable x implement the functions $x^{(0, m-1, \dots, m-1)} x^{(0, 0, m-1, \dots, m-1)}$,

$\dots x^{(0, 0, \dots, 0, m-1)}$ and use them as follows. For each n_a -tuple first described realize the product $x_1^{(B_{11}, B_{12}, \dots, B_{1m})} x_2^{(B_{21}, B_{22}, \dots, B_{2m})}$, $\dots x_{n_a}^{(B_{n_a 1}, B_{n_a 2}, \dots, B_{n_a m})}$ where

$$B_{ij} = 0 \text{ if } j < x_i$$

$$B_{ij} = m-1 \text{ if } j \geq x_i$$

The function $T(w_a \geq i)$ is the maximum of all the product terms thus obtained. Example: A checker for a code of length 3, weight 3 in a 3-valued system. The code is {012, 102, 021, 111, 201, 120, 210}. Let A be the first 2 digits x_1 and x_2 and B be x_3 .

(B_{11}, B_{12}, B_{13})			(B_{21}, B_{22}, B_{23})		
A	w_a	x_1	B	w_b	x_3
01	1	$x_1^{(222)} \cdot x_2^{(022)} \cdot x_2^{(022)} \cdot x_2^{(022)}$	0	0	$x_3^{(222)} = 2$
10	1	$x_1^{(022)}$	1	1	$x_3^{(022)}$
11	2	$x_1^{(022)} \cdot x_2^{(022)}$	2	2	$x_3^{(002)}$
02	2	$x_2^{(002)}$			
20	2	$x_1^{(002)}$			
12	3	$x_1^{(022)} \cdot x_2^{(002)}$			
21	3	$x_1^{(002)} \cdot x_2^{(022)}$			

$$f = T(w_a \geq 0) \cdot T(w_b \geq 1) + T(w_a \geq 2) \cdot T(w_b \geq 1)$$

$$= [x_1^{(022)} \cdot x_2^{(022)} + x_1^{(002)} \cdot x_2^{(002)} + x_1^{(002)} \cdot x_2^{(022)} + x_1^{(022)} \cdot x_2^{(002)}] \cdot x_3^{(022)}$$

$$g = T(w_a \geq 1) \cdot T(w_b \geq 2) + T(w_a \geq 3) \cdot T(w_b \geq 0)$$

$$= (x_1^{(022)} + x_2^{(022)}) \cdot x_3^{(002)} + (x_1^{(002)} \cdot x_2^{(022)} + x_1^{(022)} \cdot x_2^{(002)}) \cdot 2$$

The absorption laws hold so $x^{(002)} + x^{(002)} \cdot y^{(022)} = x^{(002)}$ and $2 \cdot x = x$, hence

$$f = (x_1^{(002)} + x_2^{(002)} + x_1^{(022)} \cdot x_2^{(022)}) \cdot x_3^{(022)}$$

and

$$g = (x_1^{(022)} + x_2^{(022)}) \cdot x_3^{(002)} + x_1^{(002)} \cdot x_2^{(022)} + x_1^{(022)} \cdot x_2^{(002)}$$

This can be implemented using 6 unary gates, 5 MAX gates and 5 MIN gates.

B. SFS Checkers for Two-rail Codes

Checkers for two-rail codes have for an input code space a product of basic two-rail codes and produce a single two-rail coded output. Let k be the degree of the product of two-rail input code spaces. Let a_k and b_k be the k^{th} two-rail input. The output functions f_k and g_k can be expressed recursively as follows:

$$f_k = f_{k-1} \cdot b_k + g_{k-1} \cdot a_k$$

$$g_k = f_{k-1} \cdot a_k + g_{k-1} \cdot b_k \quad \text{where } f_1 = a_1 \text{ and } g_1 = b_1$$

We shall not prove this result. However, it should be apparent from the example below that these formulae work in general.

Example: For $k=2$, the functions become

$$f_2 = a_1 b_2 + b_1 a_2$$

$$g_2 = a_1 a_2 + b_1 b_2$$

The checker's outputs f_2 and g_2 for code space inputs are as follows:

a_1	b_1	a_2	b_2	f_2	g_2
0	2	0	2	2	0
0	2	2	0	0	2
0	2	1	1	1	1
2	0	0	2	0	2
2	0	2	0	2	0
2	0	1	1	1	1
1	1	0	2	1	1
1	1	2	0	1	1
1	1	1	1	1	1

It can be verified that the code disjoint property is satisfied. Checkers for two rail codes with larger k 's can be obtained by cascading in a tree fashion the two-rail checkers for $k=2$ hence suggesting a proof to the generalized expression for f_k and g_k . The two-rail checker for $k=2$ is shown in Fig. 3.

This is a MIN-MAX implementation. A MAX-MIN implementation can be obtained from the following formulae:

$$f_k = (f_{k-1} + a_k) \cdot (g_{k-1} + b_k)$$

$$g_k = (f_{k-1} + b_k) \cdot (g_{k-1} + a_k) \quad \text{with}$$

$$f_1 = a_1 \text{ and } g_1 = b_1$$

The two-rail checkers can be used as duplication comparators if one set of inputs (e.g. all a_i

or all b_i) is inverted using $x^{(m-1, m-2, \dots, 1, 0)}$. This unary operator, however, is negative unate, hence some of the structural properties are lost and the checker is no longer SFS with respect to F_u .

C. SFS Checkers for Berger Codes

Checkers for separable unordered codes under the structural constraints enunciated before possess little well-established theory and implementation methods. This is true for any valued system. Figure 4 gives a schematic of how checking could be achieved. It is highly likely that such an approach would result in a non-unate implementation. However, it may be the only practical implementation. Other ideas can be borrowed from [7,8].

D. General Comments

It is apparent from the previous discussions on checkers that there are several open questions concerning checkers for multi-valued unordered codes. For the fixed-weight codes it would be desirable to be able to determine the existence and the construction rules for checkers with the smallest number of levels. Also modular checkers would enhance the practicability of the codes. Clearly the results on Berger checkers need further development. These topics will be discussed in a subsequent paper.

VI. Conclusions

This paper presents a scheme for decreasing undetected errors that can be produced by faulty multiple-valued combinational logic networks. It is based on a fault model that we believe is much more realistic than the more common stuck-at model. An added feature that we feel is very important is the protection provided against intermittent faults.

Fault secure networks depend on encoded inputs and outputs, and in multiple-valued technologies where pin counts will be of less importance than in current binary technologies, these encodings should cost relatively little in relation to the savings gained from added output reliability.

Constraints on network structure are relatively loose. Future study will be directed at specific network realizations. Emphasis will be placed on more common functional blocks such as adders and check circuits.

References

- [1] Su, S. Y. H., "Symposium Chairman's Message," Proceedings of Sixth International Symposium on Multiple-Valued Logic, p. i May 1976.
- [2] Von Neumann, J., "Probabilistic Logics and Synthesis of Reliable Organisms from Unreliable Components," Automata Studies, Annals of Math. Studies No. 34, C. E. Shannon and J. McCarthy, eds., Princeton University Press, Princeton, NJ, pp. 43-98, 1956.
- [3] Anderson, D. A., "Design of Self-Checking Digital Networks Using Coding Techniques," Coordinated Science Laboratory Report R-527, University of Illinois, Sept. 1971.
- [4] Carter, W. C., and P. R. Schneider, "Design of Dynamically Checked Computers," IFIP 68, vol. 2, Edinburgh, Scotland, pp. 878-883, Aug. 1968.
- [5] Smith, J. E., and G. Metze, "Strongly Fault Secure Logic Networks," to appear IEEE Trans. on Computers, June 1978.
- [6] Dussault, J., "On the Design of Self-Checking Systems under Various Fault Models," Coordinated Science Laboratory Report R-791, University of Illinois, October 1977.
- [7] Smith, J. E., "The Design of Totally Self-Checking Check Circuits for a Class of Unordered Codes," Journal of Design Automation and Fault-Tolerant Computing, vol. 1, pp. 321-342, October 1977.
- [8] Ashjaee, M. J., "Totally Self-Checking Check Circuits for Separable Codes," Ph.D Thesis, University of Iowa, July 1976.
- [9] Breuer, M. A., and A. D. Friedman, Diagnosis and Reliable Design of Digital Systems, Computer Science Press, 1976.
- [10] Shooman, M. L., Probabilistic Reliability: an Engineering Approach, McGraw-Hill, 1968.
- [11] Reddy, S. M., "A Note on Self-Checking Checkers," IEEE Trans. on Computers, vol. C-23, pp. 1100-1102, Oct. 1974.
- [12] Anderson, D. A., and G. Metze, "Design of Totally Self-Checking Check Circuits for m-out-of-n Codes," IEEE Trans. on Computers, vol. C-22, pp. 263-269, March 1973.
- [13] Diaz, M., "Design of Totally Self-Checking and Fail-Safe Sequential Machines," Digest of the Fourth Annual Symposium on Fault-Tolerant Computing, pp. 3-19 to 3-24, June 1974.
- [14] Smith, J. E., "The Design of Totally Self-Checking Combinational Circuits," Coordinated Science Laboratory Report R-737, University of Illinois, Aug. 1976.
- [15] Armstrong, D. B., "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Networks," IEEE Trans. on Electronic Computers, vol. EC-15, pp. 66-73, Feb. 1966.
- [16] Post, E. L., "Introduction to a General Theory of Elementary Propositions," American Journal of Math., vol. 43, pp. 163-185, 1921.
- [17] Rosenbloom, P. C., "Post Algebras I. Postulates and General Theory," American Journal of Math., vol. 64, pp. 167-188, 1942.
- [18] Lubell, D., "A Short Proof of Sperner's Lemma," Journal of Comb. Theory, vol. 1, p. 299, Sept. 1966.
- [19] Berger, J. M., "A Note on Error Detection Codes for Asymmetric Channels," Information and Control, vol. 4, pp. 68-73, March 1961.

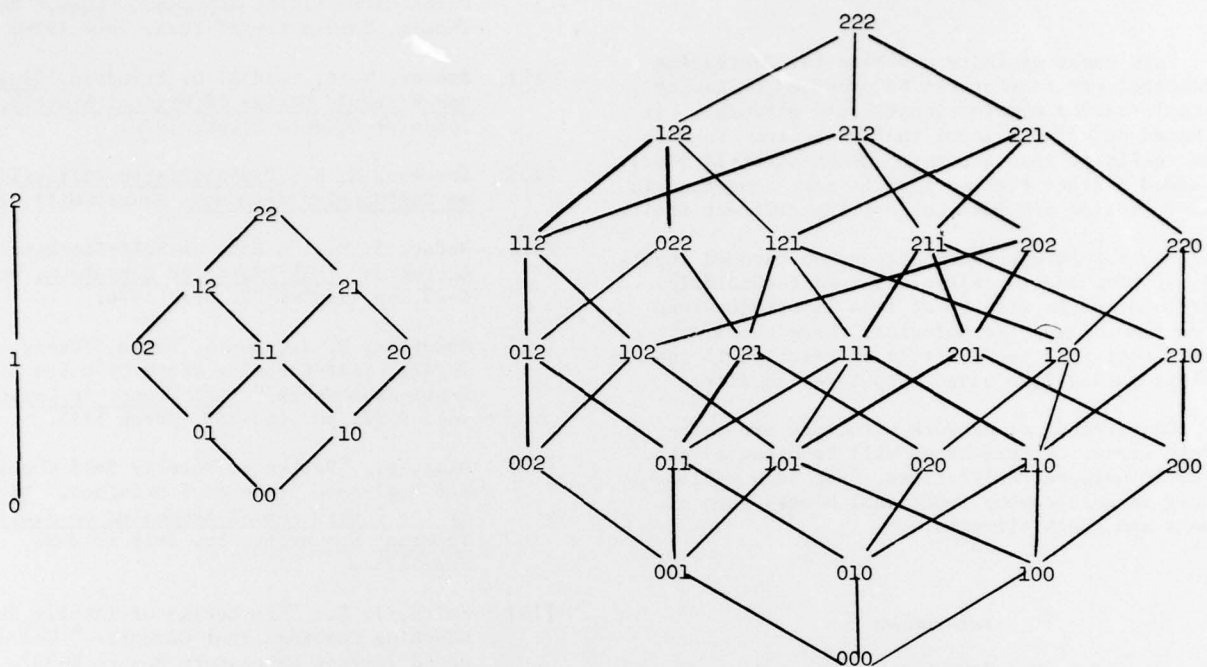


Figure 1. The lattices p , $p \times p$, $p \times p \times p$.

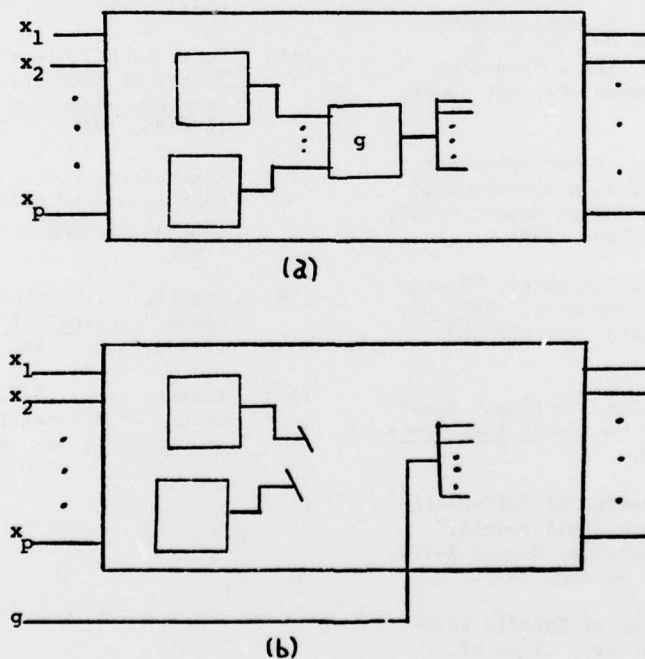


Figure 2. a) A functional block G with faulty gate g .
b) The block G' with the output of g treated as an input.

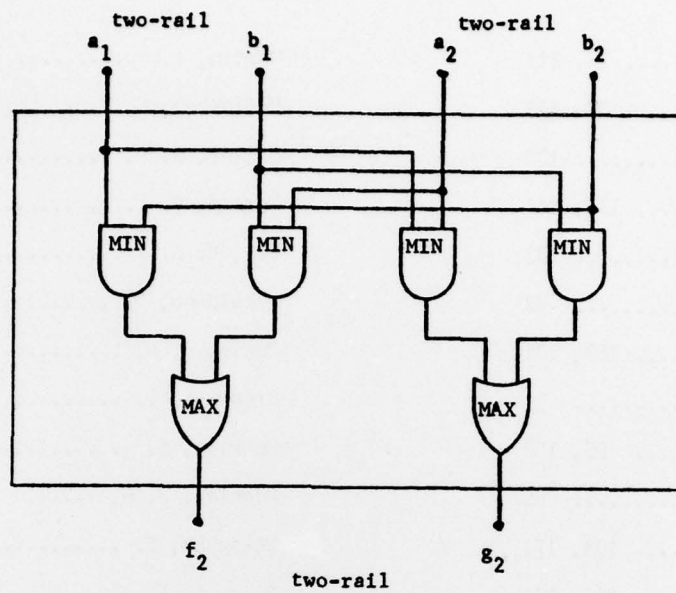


Figure 3. An SFS two-rail checker.

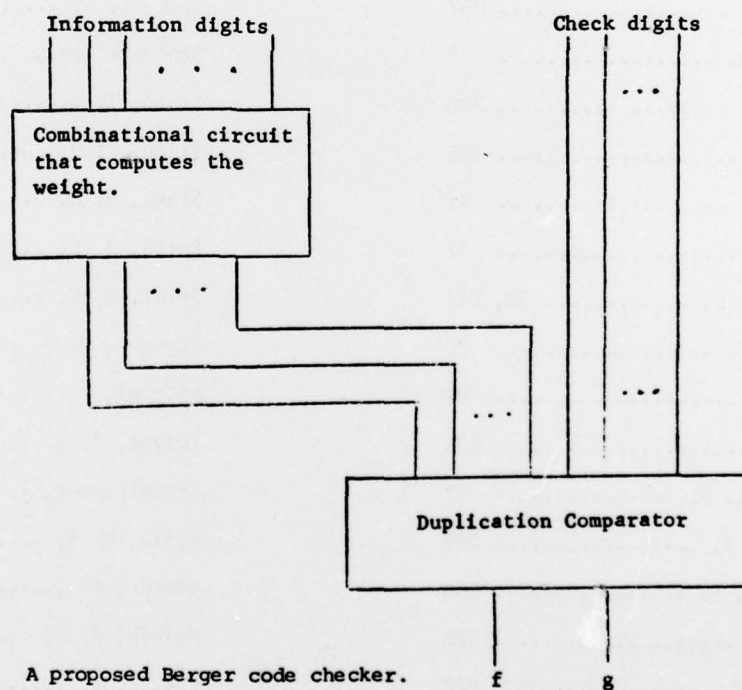


Figure 4. A proposed Berger code checker.

AUTHOR INDEX

Abdul-Karin, M. A. H.	73, 221	Martin, L.	142
Acha, J. I.	32, 213	McCluskey, E. J.	14
Ajabnoor, Y. M.	128	Miller, J. P.	157
Armstrong, J. P.	101, 114	Moraga, C.	149
Atkins, D.	33	Now, D. A.	95, 187
Berbat, H. E.	73	Mukaidono, M.	269
Breuer, M. A.	163, 202	Muzio, J. C.	235
Crist, S. C.	1	Ninomiya, T.	195
Current, K. W.	95, 187	Perrine, S.	259
Dao, T. T.	55	Pugsley, J. H.	23
Davio, M.	104, 171	Reischer, C.	142
Deschamps, J. P.	104, 171	Rine, D. C.	276
Dussault, J.	287	Rosenberg, I. C.	142
Epstein, C.	257	Sack, I. H.	242
Etiemble, D.	7	Sanchez-Comez, C.	213
Fricke, J.	208	Sasan, T.	65
Goto, M.	195	Silio, C. B., Jr.	23
Higuchi, T.	47	Singh, A. D.	114
Hoshi, H.	47	Smith, J. E.	287
Huertas, J. L.	32, 213	Srini, V. P.	188
Iturrioz, L.	76	Steward, H. H.	122
Kandel, A.	87	Stickel, M.	91
Kao, S.	195	Thayse, A.	171
Kauffman, L. H.	82	Messelkamper, T. C.	235
Kohout, L. J.	260	Wills, M. S.	226
Krolikoski, S. J.	258	Winker, S.	251
Lee, S. C.	128	Wojcik, A. S.	179
Levendel, Y.	163, 202	Wos, L.	251
Macías, L.	32	Yang, T. C.	179

PUBLICATIONS ORDER FORM

If you can't attend the Eighth International Symposium on Multiple-Valued Logic but would like the proceedings shipped directly to you, simply fill out and return the order form below with your check or money order payable to the IEEE Computer Society

Mail to: IEEE Computer Society
5855 Naples Plaza, Suite 301
Long Beach, CA 90803

I am unable to attend the Eighth International Symposium on Multiple-Valued Logic. Please ship me the proceedings indicated below (check appropriate boxes)

	Non-Member	Member	Catalog No.
Eighth International Symposium on Multiple Valued Logic — 1978	<input type="checkbox"/> \$20.00	<input type="checkbox"/> \$15.00	
Seventh International Symposium on Multiple Valued Logic 1977	<input type="checkbox"/> \$20.00	<input type="checkbox"/> \$15.00	
Sixth International Symposium on Multiple Valued Logic 1976	<input type="checkbox"/> \$20.00	<input type="checkbox"/> \$15.00	006
Fifth International Symposium on Multiple Valued Logic 1975	<input type="checkbox"/> \$20.00	<input type="checkbox"/> \$15.00	003
Fourth International Symposium on Multiple Valued Logic 1974	<input type="checkbox"/> \$20.00	<input type="checkbox"/> \$15.00	009
Third International Symposium on Multiple Valued Logic 1973	<input type="checkbox"/> \$20.00	<input type="checkbox"/> \$15.00	013

(CHECK ONE)

- ☐ MY PAYMENT OF _____ IS ENCLOSED.
- ☐ ADD \$2.00 BILLING CHARGE AND INVOICE Me. (California residents add 6% sales tax.)

Name _____

*Computer Society or
IEEE Member No. _____

Address _____

City _____

State _____

Zip _____



Non-Profit Organization
U. S. Postage Paid
Silver Spring, Md.
20901
Permit No. 1398